The Dissertation Committee for Young-ri Choi
certifies that this is the approved version of the following dissertation:

Design and Analysis of Self-Stabilizing Sensor Network Protocols

Committee:

Mohamed G. Gouda, Supervisor

Simon S. Lam

Aloysius K. Mok

Lili Qiu

Anish Arora

Design and Analysis of Self-Stabilizing Sensor Network Protocols

by

Young-ri Choi, B.S.; M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2007

Dedicated to my husband and my parents.

Acknowledgments

I would like to express my thanks to many people for their help and support. First of all, I would like to thank my advisor, Dr. Mohamed G. Gouda, for his help and guidance. He taught me how to design elegant network protocols, and he always encouraged me to work on new problems. I am also grateful to my committee members, Dr. Simon S. Lam, Dr. Aloysius K. Mok, Dr. Lili Qiu, and Dr. Anish Arora for their valuable comments and time.

I would like to thank Dr. Ted Herman, Dr. Sandeep Kulkarni, Dr. Rajiv Ramnath, and Dr. Mikhail Nesterenko. They gave me valuable input regarding my work on two DARPA NEST field demonstrations. I also would like to thank my fellow graduate students who worked together for these demonstrations, Hongwei Zhang, Sandip Bapat, Vinodkrishnan Kulathumani, Vinayak Naik, Limin Wang, and Mahesh Arumugam.

I am deeply grateful to my parents for their love and support. At last, I would like to thank my husband, Jaehyuk Huh, who is also my best friend. It would be impossible for me to complete this dissertation without his endless support and encouragement.

Design and Analysis of Self-Stabilizing Sensor Network Protocols

Publication No. _____

Young-ri Choi, Ph.D.
The University of Texas at Austin, 2007

Supervisor: Mohamed G. Gouda

A sensor is a battery-operated small computer with an antenna and a sensing board that can sense magnetism, sound, heat, etc. Sensors in a network communicate and cooperate with other sensors to perform given tasks. A sensor network is exposed to various dynamic factors and faults, such as topology changes, energy saving features, unreliable communication, and hardware/software failures. Thus, protocols in this sensor network should be able to adapt to dynamic factors and recover from faults.

In this dissertation, we focus on designing and analyzing a class of sensor network protocols, called self-stabilizing protocols. A self-stabilizing protocol is guaranteed to return to a state where it performs its intended function correctly, when some dynamic factors or faults corrupt the state of the protocol arbitrarily. Therefore, in order to make a sensor network resilient

to dynamic factors and faults, each protocol in the sensor network should be self-stabilizing.

We first develop a state-based model that can be used to formally specify sensor network protocols. This model accommodates several unique characteristics of sensor networks, such as unavoidable local broadcast, probabilistic message transmission, asymmetric communication, message collision, and timeout actions and randomization steps. Second, we present analysis methods for verifying and analyzing the correctness and self-stabilization properties of sensor network protocols specified in this model. Third, using the state-based model and analysis methods, we design three self-stabilizing sensor network protocols, prove their self-stabilization properties, and estimate their performance. These three self-stabilizing protocols are a sentry-sleeper protocol that elects a sentry from a group of sensors at the beginning of each time period, a logical grid routing protocol that builds a routing tree whose root is the base station, and a family of flood sequencing protocols that distinguish between fresh and redundant flood messages using sequence numbers.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

A sensor is a battery-operated small computer with an antenna and a sensing board that can sense magnetism, sound, heat, etc. Sensors in a network can use their antennas to communicate in a wireless fashion by broadcasting messages over radio frequency to neighboring sensors in the network. Due to the limited range of radio transmission, sensor networks are usually multi-hop. Sensor networks can be used for military, environmental, and commercial applications such as intrusion detection [4, 7], habitat monitoring [49, 75], micro-climatic monitoring [66], and equipment health monitoring [42].

A sensor network is exposed to various dynamic factors and faults, such as topology changes, energy saving features, unreliable communication, and hardware/software failures. These dynamic factors and faults can cause a protocol executed in the network to reach *an illegitimate state*, where the protocol cannot perform its intended function correctly. A protocol is called *self-stabilizing* if starting from any illegitimate state, the protocol eventually converges to *a legitimate state*, and once it reaches a legitimate state, it continues to perform its intended function correctly [5, 23]. Therefore, in order to make a sensor network resilient to dynamic factors and faults, each protocol

in the network should be self-stabilizing.

A self-stabilizing protocol provides a high degree of fault-tolerance for transient faults that can corrupt the state of the protocol arbitrarily. Examples of transient faults in a sensor network are memory corruption, message corruption, and sensor failure and recovery. We assume that these faults do not continuously occur in the network. A self-stabilizing sensor network protocol recovers from these transient faults without any manual intervention.

Developing self-stabilizing sensor network protocols is a challenging task, since sensor networks have several unique characteristics, such as unavoidable local broadcast, probabilistic message transmission, asymmetric communication, message collision, and timeout actions and randomization steps. These characteristics should be taken into account when one designs or analyzes self-stabilizing sensor network protocols.

Self-stabilization has been studied in the literature of distributed systems [5, 6, 21, 23, 58]. However, the models used in the literature (such as shared memory models) do not accommodate the unique characteristics and specific behaviors of sensor networks. Thus, self-stabilizing protocols that are developed based on these models are not suitable for sensor networks.

Several self-stabilizing sensor network protocols have been developed based on abstract models. In these models, communications between sensors are abstracted via shared variables [19], and message channels [12, 20], which, we believe, are not suitable for designing and analyzing sensor network proto-

cols.

There have been some efforts on transformations of self-stabilizing protocols from the models of distributed systems to a sensor network model, where local broadcast and message collision are taken into account in shaping the execution of sensor networks [34, 43]. However, these models assume symmetric communications between sensors, and do not consider probabilistic message transmission, and timeout actions and randomization steps, which are very common in practical sensor networks.

1.1 Contributions

In this dissertation, we focus on designing and analyzing self-stabilizing sensor network protocols. This dissertation has three major contributions as follows. First, we develop a state-based model that can be used to formally specify sensor network protocols (in short, sensor protocols). Second, we present analysis methods for verifying and analyzing the correctness and self-stabilization properties of sensor protocols specified in this model. Third, we design and analyze several self-stabilizing sensor protocols that play critical roles in sensor networks. For each of these protocols, the protocol is formally specified using our state-based model, and the self-stabilization properties of the protocol and its performance are verified and analyzed using our analysis methods. Next, we discuss each of our contributions in detail.

1.1.1 A State-based Model of Sensor Protocols

The unique characteristics of sensor networks, such as unavoidable local broadcast, probabilistic message transmission, asymmetric communication, message collision, and timeout actions and randomization steps, make sensor protocols hard to specify formally and even harder to verify or analyze. Moreover, these characteristics distinguish sensor networks from other distributed systems, and make existing models for distributed systems unsuitable for sensor networks. As a result, in most previous work, sensor protocols have been developed and implemented without formal descriptions of the protocols, and usually tested by simulation, which cannot guarantee the correctness of the protocols.

To solve these problems, we develop a state-based model of sensor protocols, which accommodates the above characteristics of sensor networks [27]. This model allows us to formally specify and verify the desired properties of sensor protocols, and also allows us to develop the simulation of the protocols so that we can estimate their performance.

1.1.2 Analysis Methods of Sensor Protocols

The model of sensor protocols presented in this dissertation is rather detailed. Thus, we propose three methods for analyzing a sensor protocol specification based on this model [27].

In the first method, called *nondeterministic analysis*, a specified sensor protocol is shown to be "nondeterministically correct" under the assumption

that message delivery is assured and message collision is guaranteed not to occur. In the second method, called *probabilistic analysis*, the protocol is shown to be "probabilistically correct" under the assumption that message delivery is probabilistic but message collision is guaranteed not to occur. In the third method, called *collision analysis*, the effect of message collision and probabilistic message delivery on the correctness of the protocol is analyzed, and appropriate values for some parameters in the protocol are chosen to achieve a compromise between two conflicting factors: make the probability of message collision reasonably small, and make the protocol reach a target state within a reasonably short time.

To demonstrate the utility of our model and analysis methods, we discuss an example protocol that can be used by a sensor to identify its strong neighbors in the network, and apply the three analysis methods, mentioned above, to the protocol to analyze its correctness.

1.1.3 Sentries and Sleepers in Sensor Networks

One of the challenging problems in designing sensor networks is to lengthen the lifetime of sensor batteries. To solve this problem, we can replace each sensor in a network with a group of sensors and make the group of sensors act as one sensor.

We present a sentry-sleeper protocol that elects one sensor from a group of sensors at the beginning of each time period [28]. This protocol can be used to lengthen the lifetime of the group by making the elected sensor, called

a sentry, stay awake and the other sensors, called sleepers, go to sleep. This protocol is self-stabilizing such that starting from any state, the protocol eventually converges to a legitimate state where exactly one sensor in the group is a sentry. We show by analysis and simulation that by adopting this protocol, a group of n sensors can lengthen its lifetime almost n times the lifetime of a single sensor. This protocol does not require sensors in a group to have unique identifiers. This feature makes our protocol resilient against any attack by an adversary sensor in the group that may lie about its own identifier to be elected a sentry over and over, and keep the legitimate sensors in the group asleep for a long time.

1.1.4 Logical Grid Routing in Sensor Networks

It is difficult to design a routing protocol for sensor networks, since such a protocol should overcome the special challenges of sensor networks. First, the protocol should not consume a large fraction of the network resources. Second, the protocol should be able to recover from various faults such as memory corruption, and sensor failure and recovery. Third, the protocol should avoid unreliable long links in sensor networks that prevent building a reliable routing tree.

We develop a routing protocol, called the logical grid routing protocol, that builds an incoming spanning tree whose root is the base station of a sensor network [17]. This routing protocol is stabilizing such that starting from any state, the protocol converges to a legitimate state where all the sensors that can

be connected to the routing tree are connected to the tree. The convergence time of the protocol is proportional to the diameter of the sensor network. This routing protocol also has several other advantages over earlier protocols: overhead of the protocol is small, the protocol avoids unreliable long links to build a reliable routing tree, the protocol balances the load over the whole network, and it has nice fault-tolerance property. We show by experiments that this protocol provides reliable message delivery to the base station under heavy and bursty traffic.

1.1.5 Flood Sequencing Protocols in Sensor Networks

Flood is a communication primitive that can be used by the base station of a sensor network to send a copy of a message to every sensor in the network. When a sensor receives a flood message, the sensor needs to check whether it has received this message for the first time or not. If the sensor has received this message for the first time, the message is fresh, and so the sensor accepts it and may forward it to its neighbors. Otherwise, the message is redundant and so the sensor discards it. We call a protocol that uses sequence numbers to distinguish between fresh and redundant flood messages a *flood sequencing protocol*. A flood sequencing protocol should be designed such that when some fault corrupts the state of the protocol arbitrarily, the protocol eventually converges to a legitimate state where every sensor accepts every fresh message and discards every redundant message.

We discuss a family of four flood sequencing protocols. They are a

sequencing free protocol, a *linear sequencing* protocol, a *circular sequencing* protocol, and a *differentiated sequencing* protocol. We also analyze the stabilization properties of these four protocols, and compare them against each other. We conclude that the differentiated sequencing protocol has better stabilization property and provides better performance than those of the other three protocols.

1.2 Organization

The organization of this dissertation is as follows. In Chapter 2, we introduce a state-based model of sensor network protocols. In Chapter 3, we present three analysis methods for sensor network protocols. In the following three chapters, we specify and analyze three self-stabilizing sensor network protocols. In Chapter 4, a sentry-sleeper protocol is discussed. In Chapter 5, a logical grid routing protocol is presented. In Chapter 6, a family of flood sequencing protocols is discussed. We finally make concluding remarks in Chapter 7.

Chapter 2

A State-based Model of Sensor Protocols

Sensor networks and their protocols have several characteristics that make them hard to specify formally and even harder to verify. Examples of these characteristics are

i. *Unavoidable local broadcast:* When a sensor sends a message, even one that is intended for a particular neighboring sensor, a copy of the message is received by every neighboring sensor.

ii. *Probabilistic message transmission:* When a sensor sends a message, the message reaches the different neighboring sensors (and can be received by each of them) with different probabilities.

iii. *Asymmetric communication:* Let u and v be two neighboring sensors in a network. The probability that a message sent by u is received by v can be different from the probability that a message sent by v is received by u.

iv. *Message collision:* If two neighboring sensors send messages at the same time, then neither sensor receives the message from the other sensor. Moreover, if two (not necessarily neighboring) sensors send messages at

the same time, then any sensor that is a neighbor of both sensors will not receive any of the two messages. In this case, the two messages are said to have collided.

v. *Timeout actions and randomization steps:* Given the above characteristics of a sensor network, it seems logical that sensor protocols need to heavily depend on timeout actions and randomization steps to perform their functions.

The above characteristics of sensor protocols are far from common in the literature of distributed systems. Thus, one is inclined to believe that the "standard model" of distributed systems is not suitable for sensor protocols. The search for a suitable model for sensor protocols is an obligatory first step towards formal specification, verification, and design of these protocols.

There have been earlier efforts to model the software of sensor networks. Examples of these efforts are [47], [3], [54], and [48]. We review these and other efforts in the related work section of this chapter. Nevertheless, it is important to state here that all these efforts are not directed towards modeling sensor protocols; rather they are directed toward modeling sensor network applications. Clearly, sensor protocols are quite different from sensor applications in terms of their functions and in terms of how they accomplish these functions. For instance, sensor protocols need to deal with the intricate characteristics of sensor networks, as they attempt to hide these characteristics from the sensor applications. Thus, whereas a sensor protocol has to deal with unavoidable

local broadcast, probabilistic message transmission, asymmetric communication, and message collision, a sensor application can view the sensor network as a reliable medium for communicating sensing data. Also whereas a sensor protocol depends heavily on timeout actions and randomization steps, a sensor application rarely needs to resort to these devices.

2.1 Topology of Sensor Networks

The *topology* of a sensor network is a directed graph where each node represents a distinct sensor in the network and where each directed edge is labeled with some probability. A directed edge (u,v), from a sensor u to a sensor v, that is labeled with probability p, where $p > 0$, indicates that if sensor u sends a message, then this message arrives at sensor v with probability p (provided that neither sensor v nor any "neighboring sensor" of v sends another message at the same time).

There are two probabilities, α and β, where $\alpha > \beta$, that label the edges in the topology of a sensor network. An edge that is labeled with the large probability α is called a *strong edge*, and an edge that is labeled with the small probability β is called a *weak edge*. In this chapter, we assign α 0.95 and β 0.5. Below we discuss some experiments that we have carried out on sensors and led us to this choice of probabilities in the topology of a sensor network [18]. It is important to note that although the probability labels of α and β are chosen to be 0.95 and 0.5 in this chapter, different values can be chosen for these probability labels for different setting of sensor networks.

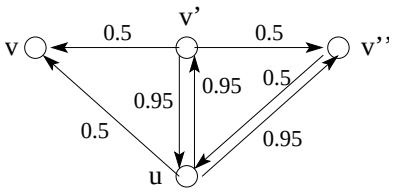


Figure 2.1: Topology of a sensor network

Let u and v be two distinct sensors in a network. Sensors u and v are called *strong neighbors* iff there are two strong edges between them in the network topology. The two sensors are called *middle neighbors* iff there are one strong edge and one weak edge between them in the network topology. Sensors u and v are called *weak neighbors* iff there is exactly one edge between them, or there are two weak edges between them in the network topology. They are called *non-neighbors* iff there are no edges between them in the network topology. If there is an edge from u to v in the network topology, then u is called an *in-neighbor* of v and v is called an *out-neighbor* of u.

As an example, Fig. 2.1 shows the topology of a sensor network that has four sensors. In this network, sensors u and v are weak neighbors, sensors u and v' are strong neighbors, sensors u and v'' are middle neighbors, and sensors v and v'' are non-neighbors. Sensor u has three out-neighbors, namely sensors v, v', and v''. Thus, if sensor u sends a message, this message is received by each of the two sensors v' and v'' with probability 0.95 and is received by sensor v with probability 0.5. Also sensor u has two in-neighbors, namely sensors v', and v''. Thus, if sensor v' sends a message, then sensor u receives this message with probability 0.95, and if sensor v'' sends a message, then sensor u receives

this message with probability 0.5.

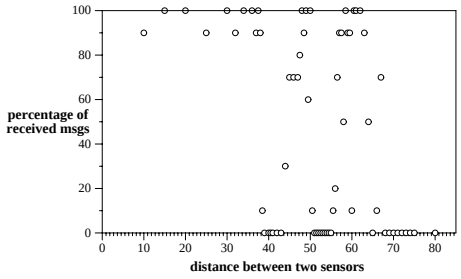


Figure 2.2: Percentage of received messages

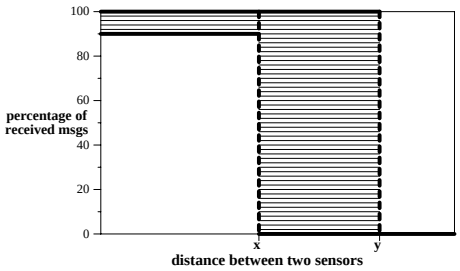


Figure 2.3: Idealized percentage of received messages

In [18], we describe some experiments that we have carried out using Mica sensors [1]. In these experiments, a sensor u sends a sequence of messages at the rate of one message per 5 seconds, and another sensor v attempts to receive all the sent messages. The results of these experiments are summarized in Fig. 2.2 where each point represents the result of one experiment. (Similar results are also reported in [68] and [14].)

We observe that from Fig. 2.2 if the distance between two sensors u and v is in the range 0 .. 38 inches, v receives between 90% and 100% of the messages sent by u. On the other hand, if the distance between sensors u and

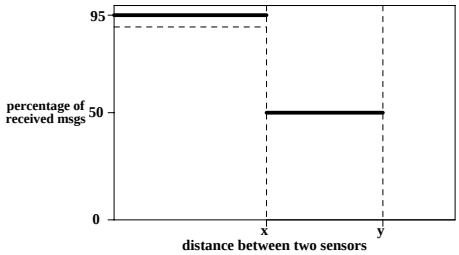


Figure 2.4: The probability label of an edge

v is in the range 38 .. 67 inches, v receives anywhere between 0% and 100% of the messages sent by u. Finally, the distance between sensors u and v is longer than 67 inches, v receives 0% of the messages sent by u. From these observations, the diagram in Fig. 2.2 can be "idealized" as shown in Fig. 2.3.

Let u and v be distinct sensors in the topology of a sensor network, and assume there is a directed edge from u to v in the network topology. According to the idealized diagram in Fig. 2.3, if the distance between u and v is in the range 0 .. x, then v receives between 90% and 100% of the messages sent by u. Thus, the directed edge from u to v can be labeled with a probability 0.95, and the edge is strong. If the distance between u and v is in the range x .. y, then v receives between 0% and 100% of the messages sent by u. Thus, the directed edge from u to v can be labeled with a probability 0.5, and the edge is weak. Fig. 2.4 shows how the probability label of an edge from one sensor to another in the network topology is chosen based on the distance between the two sensors.

2.2 Sensor Network Execution

A sensor is specified as a program that has global constants, local variables, and one or more actions. In general, a sensor is specified as follows:

```
sensor <sensor name>

const <const name> : <const type>, ... , <const name> : <const type>
var   <var name> : <var type>, ... , <var name> : <var type>

begin
   timeout-expires  -> <action statements>       // timeout action
[] rcv <msg.0>      -> <action statements>       // receiving action
   ...                 ...
[] rcv <msg.k-1>    -> <action statements>       // receiving action
end
```

Note that the actions of a sensor consist of exactly one timeout action and zero or more receiving actions. Before we can discuss the execution of sensor actions, we need to explain our model of real-time.

We assume that the real-time passes through discrete time instants: instant 1, instant 2, instant 3, and so on. The time periods between consecutive instants are equal. Executions of the different actions of a sensor occur only at the time instants, and not during the time periods between instants. We refer to the time period between two consecutive instants t and $t + 1$ as the *time unit* $(t, t + 1)$. (The value of a time unit is not critical to the current presentation, but we estimate that the value of the time unit is around 100 milliseconds.)

At a time instant t, if the timeout of a sensor u expires, then u executes its timeout action (at t). Executing the timeout action of sensor u at t causes

u to update its local variables, and to send at most one message at t. It also causes u to execute the statement "timeout-after <expression>" which causes the timeout of u to expire (again) after k time units, where k is the value of <expression> at the time unit $(t, t+1)$. The timeout action of sensor u is of the following form:

```
timeout-expires -> <update local variables of u>;
                   <send at most one message>;
                   <execute timeout-after <expression>>
```

To keep track of its timeout, each sensor u has an implicit variable named "timer.u". In each time unit between two consecutive instants, timer.u has a fixed positive integer value. The value of "timer.u" is determined by the following two rules:

i. If the value of timer.u is k, where $k > 1$, in a time unit $(t-1, t)$, then the value of timer.u is $k-1$ in the time unit $(t, t+1)$.

ii. If the value of timer.u is 1 in a time unit $(t-1, t)$, then sensor u executes its timeout action at instant t. Moreover, since sensor u executes the statement "timeout-after <expression>" as part of executing its timeout action, the value of timer.u in the time unit $(t, t+1)$ is the value of <expression> in the same time unit.

If a sensor u executes its timeout action and sends a message at an instant t, then an out-neighbor v of u receives a copy of the message at t, provided that the following three conditions hold.

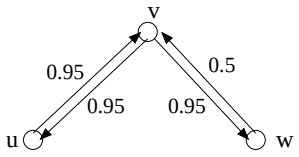


Figure 2.5: Topology of a sensor network

i. A random integer number is uniformly selected in the range 0 .. 99, and this selected number is less than $100 * p$, where p is the probability label of edge (u,v) in the network topology.

ii. Sensor v does not send any message at instant t.

iii. For each in-neighbor w of v, other than u, if w sends a message at t, then a random integer number is uniformly selected in the range 0 .. 99, and this selected number is at least $100 * p'$, where p' is the probability label of edge (w,v) in the network topology.

As an example, Fig. 2.5 shows the topology of a sensor network that consists of three sensors, u, v, and w. Assume that sensor u sends a message m at instant t. To determine whether message m will be received at sensor v (at t), we need to consider three cases:

i. *Sensor v sends a message m' at t:* In this case, v does not receive message m.

ii. *Neither sensor v nor sensor w sends any message at t:* In this case, a random number is uniformly selected in the range 0 .. 99. If this number

is less than 95, then sensor v receives message m at t. If this number is at least 95, then sensor v does not receive message m.

iii. *Sensor v does not send any message, but sensor w sends a message m' at t:* In this case, two random numbers, n and n', are uniformly selected in the range 0 .. 99. There are four cases to consider.

   (a) $n < 95$ and $n' < 50$: m and m' *collide* and v receives no message.

   (b) $n < 95$ and $n' \geq 50$: v receives m (and m' is lost).

   (c) $n \geq 95$ and $n' < 50$: v receives m' (and m is lost).

   (d) $n \geq 95$ and $n' \geq 50$: m and m' are lost and v receives no message.

We conclude from the above example that a sensor can receive at most one message at any instant. If two or more messages are sent at the same instant by the neighboring sensors of a sensor u, then these messages collide with one another and sensor u ends up receiving no message.

If a sensor u receives a message <`msg.i`> at an instant t, then u executes the following receiving action at t.

```
rcv <msg.i> -> <update local variables of u>;
               <may execute timeout-after <expression>>
```

Note that executing the receiving action of sensor u causes u to update its own local variables. It may also cause u to execute the statement "timeout-after <expression>" which causes the timeout of u to expire after k time units,

where k is the value of <expression> in the time unit $(t,\ t+1)$. Note that executing the receiving action of sensor u does not cause u to send any message.

Let us summarize how the execution of a sensor network proceeds during one time instant t. First, the value of timer.u for every sensor u in the network is decremented by one at t. Second, if the value of any timer.u becomes 0 at t, then sensor u executes its timeout action at t. Execution of the timeout action of a sensor u at t assigns a new value to timer.u and may cause u to send one message at t. Third, if a sensor u sends a message at t, then any out-neighbor v of u may receive the message at t. Even if an out-neighbor v of u has executed its timeout action but sent no message at t, v can still receive u's message at t. In other words, a sensor may execute its timeout action followed by a receiving action at the same time instant provided that the sensor does not send a message during its execution of the timeout action. It follows from the above discussion that at a time instant, a sensor u executes exactly one of the following:

i. u sends one message, but receives no message.

ii. u receives one message, but sends no message.

iii. u sends no message and receives no message.

A *state* of a protocol is defined by a value for each variable, including the implicit variable timer.u, for each sensor u in the protocol. In every state (whether legitimate or illegitimate), a value of each variable is selected from the declared domain of the variable.

2.3 Related Work

Several models for sensor applications have been proposed [3], [54], [47], [48]. In general, the purpose of these models is to hide application programmers from low-level details such as routing, group management, resource management, etc. EnvioTrack [3] provides a high-level programming abstraction for tracking applications in sensor networks. Newton and Welsh proposed a functional language to specify the global behavior of a sensor application [54]. Liu et al. presented a state-centric programming model for sensor networks [47]. Database approach was proposed in TAG [48]. Unlike these models, our proposed model is to describe sensor protocols (that are responsible for routing, group management, etc). Thus, our model deals with the intricate characteristics of wireless sensor networks discussed at the beginning of this chapter.

A high-level and abstract model of network protocols, called the Abstract Protocol notation (AP), was proposed for traditional networks [29]. Later the Timed Abstract Protocol notation was developed, adding the ability to express temporal behavior to AP [50]. Network protocols for traditional networks (such as the Internet) can be specified and verified in these models.

Self-stabilization has been studied in the literature of distributed systems [5, 6, 21, 23, 58]. However, the models used in the literature, such as shared memory models, do not accommodate the unique characteristics and specific behaviors of sensor networks. In shared memory models, a sensor can read some variables in its neighboring sensors and write its own variables, and

so the details of message sending and receiving are hided.

Several self-stabilizing sensor protocols have been developed based on abstract models. In [19], each sensor can communicate with its neighboring sensors using shared variables. In [20], a sensor can communicate with any other sensor in the network using an abstract channel. In [12], each sensor has a FIFO message channel that can hold many messages sent by its neighbors. We believe that these models are not suitable for designing and analyzing sensor protocols. Also in [10], it is assumed that communications between neighboring sensors are reliable. This assumption is not easy to achieve in sensor networks.

There have been some efforts on transformations of self-stabilizing protocols from the models of distributed systems to a sensor network model, where local broadcast and message collision are taken into account in shaping the execution of sensor networks [34, 43]. The models are called a local broadcast model and write all with collision model. Also a similar model was used to develop a self-stabilizing TDMA slot assignment algorithm in [35]. However, these models assume symmetric communications between sensors, and do not consider probabilistic message transmission, and timeout actions and randomization steps, which are very common in practical sensor networks.

Chapter 3

Analysis Methods of Sensor Protocols

3.1 Analysis Methods

The model of sensor network protocols presented in the previous chapter is rather detailed. Thus, we propose three analysis methods for analyzing a sensor protocol specification based on this model. In the first analysis, the correctness of the protocol specification is verified under two assumptions: idealized message transmission and no message collision. In the second analysis, the protocol specification is analyzed under the relaxation of the first assumption. In the third analysis, the protocol specification is analyzed under the relaxation of the first and second assumptions.

We refer to the first analysis as *nondeterministic analysis*, to the second analysis as *probabilistic analysis*, and to the third analysis as *collision analysis*.

In the next two sections, we present an example of a sensor protocol specification, and then analyze this protocol specification using our three analysis methods.

The two assumptions, of idealized message transmission and no message collision, upon which our analysis methods are based are stated as follows.

i. *Idealized message transmission:* In the topology of a sensor network,

the probability label of each strong edge is 1 (instead of 0.95), and the probability label of each weak edge is 0 (instead of 0.5).

ii. *No message collision:* For every two distinct sensors u and v in a sensor network, if u is a (in- or out-) neighbor of v, or if the network has a third sensor w that is an out-neighbor for both u and v, then timer.u and timer.v have distinct values at every instant during the execution of the sensor network.

Some explanations concerning these two assumptions are in order. The first assumption has the effect of removing all the weak edges from the topology of a sensor network. It also has the effect of strengthening all the strong edges in a network topology.

To explain the second assumption, recall that a sensor u can send a message only during an execution of its timeout action, and that the timeout action of sensor u can be executed at an instant t iff the value of timer.u is 1 in the time unit $(t-1,\ t)$. Thus, the assumption of no message collision ensures that any two sensors, whose messages would collide if they were sent at the same instant, are guaranteed never to send messages at the same instant during any execution of the sensor network.

In order to make the second assumption, of no message collision, more acceptable, it is recommended that each statement "timeout-after x" in a sensor u be written as "timeout-after random(x, y)" where $x > 0$ and $x \leq y$. Thus, any new value assigned to timer.u is chosen uniformly from the range x

.. y. Because the new values of the timer variables are chosen uniformly from a reasonably large range, it is unlikely that any two timer variables will ever have the same value.

Next, we describe in some detail the three analysis methods.

i. Nondeterministic analysis:

This analysis is used to verify that a protocol is guaranteed to reach, from a given initial state, a desirable target state under the two assumptions of idealized message transmission and no message collision. For this analysis, we generate a state transition diagram of the protocol. In the diagram, each protocol state has one or more outgoing edges, since the protocol is specified using randomization steps of the form "timeout-after random(x,y)". From this diagram, we can verify that the protocol nondeterministically satisfies the desired reachability property.

ii. Probabilistic analysis:

This analysis is used to verify that a protocol will reach, from a given initial state, a desirable target state with a high probability, under the assumption of no message collision. For this analysis, we generate a probabilistic state transition diagram of the protocol, where each edge in the diagram is labeled with a probability. Note that the probabilities that label the edges in the probabilistic state transition diagram are computed from the probability labels in the network topology of the protocol. From this diagram, we can verify that the protocol probabilistically satisfies

the desired reachability property.

iii. Collision analysis:

The nondeterministic and probabilistic analyses (in the first two analyses) of a protocol can be carried out without specifying the values of x and y in the randomization steps "timeout-after random(x,y)" in the protocol specification. In choosing the values of x and y in these analyses, one needs to observe two restrictions. First, the difference $y - x$ should be large enough to ensure that the probability of message collision is reasonably small (and so the nondeterministic and probabilistic analyses of the protocol are reasonably accurate). Second, the difference $y - x$ should be small enough to ensure that the protocol reaches its desirable target state in a reasonably short time. To determine the appropriate values of x and y in the randomization steps, one can simulate the protocol for many value combinations of x and y and select the most appropriate values of x and y.

Note that each of the above three analyses is different from each other on its objective, complexity, and/or result quality. Performing the nondeterministic analysis on a protocol is simpler and easier than performing the probabilistic analysis on the protocol (since the nondeterministic analysis assumes the idealized message transmission). However, the probabilistic analysis yields a more realistic result than that of the nondeterministic analysis.

3.2 A Protocol Specification Example

In this section, we use the model presented in Chapter 2 to specify a sensor protocol that can be used by any sensor in order to identify the strong neighbors of that sensor in its network. We refer to this protocol as the neighbor computation protocol. (Recall that two sensors in a network are strong neighbors iff there are two strong edges between them in the network topology.)

To identify the strong neighbors of a sensor u, sensor u sends three request messages. Whenever a sensor v receives a request message sent by sensor u, sensor v sends a reply message. If sensor u receives two or more reply messages sent by the same sensor v, then sensor u concludes that sensor v is one of its strong neighbors.

Assume that the time period between two successive request messages sent by the same sensor is fixed. Under this assumption, if two neighboring sensors u and u' start to send their request messages at the same time, then the request messages sent by u will collide with the request messages sent by u' and both u and u' may end up concluding wrongly that they have no strong neighbors. Therefore, the time period between two successive request messages should be uniformly selected from a "large enough" range $1 \,..\, x$. (In the next section, we discuss how to choose a value for x.)

If every sensor v, that receives a request message from a sensor u, sends a reply message immediately after it receives the request message, then all the

reply messages will collide with one another and u may end up receiving no reply messages. Thus, when a sensor v receives a request message from a sensor u, v should wait a random period of time before it sends a reply message. The length of this time period should be uniformly selected from the range $1 \,..\, x$.

Consider the scenario where a sensor v receives a request message from a sensor u and decides to wait for some random period before it sends a reply message to u. It is possible that before v sends its reply to u, v receives another request message from another sensor u'. In this case, v should send one reply message to both u and u'. This requires that sensor v maintains a reply set, called $rset$, that contains the identifier of every sensor u from which v has received a request message and to which v has not yet sent a corresponding reply message. At the end of the above scenario, $rset$ in sensor v has the value $\{u, u'\}$.

Note that sensors u and u' in the above scenario can be the same sensor u. Thus, $rset$ in each sensor is a multiset rather than a set. For example, at the end of the above scenario, $rset$ in sensor v has the value $\{u, u\}$.

Consider the scenario where a sensor u sends a request message and decides to wait for a random period before it sends its second request message. It is possible that before u sends its second request message, u receives a request message from another sensor u'. In this case, u should send one composite message that consists of the second request message and a reply message to sensor u'. We refer to this composite message as a request-reply message. In fact, every message in our protocol, whether a request message,

a reply message, or a request-reply message, can be viewed as a request-reply message.

Each message in the neighbor computation protocol has three fields:

```
(v,b,s)
```

The first field v is the identifier of sensor v that sent this message. The second field b has two possible values: 0 and 1. If $b = 0$, then the message is a pure reply message. If $b = 1$, then the message is either a request message or a request-reply message. The third field s is the current value of $rset$ in sensor v. Note that if the message is a pure request message, then $s =$ empty set.

Each sensor u has one constant x and eight variables as follows.

```
sensor u      // sensor u where 0=< u < n

const x    : integer
var   nghs : set {u' | 0<= u' < n},       // strong ngh set
      rcvd : array [0 .. n-1] of 0..3,    // rcvd replies
      rset : set {u' | 0<= u' < n},       // reply set
      rm   : 0..3,                        // remaining request msgs
      done : boolean,                     // computation done or not
      v    : 0..n-1,                      // received sensor id
      b    : 0..1,                        // received request bit
      s    : set {u' | 0<= u' < n}        // received reply set
```

Variable $nghs$ is the set of strong neighbors that sensor u needs to compute periodically. An element $rcvd[v]$ in variable $rcvd$ contains the number of replies that sensor u has received from sensor v after u has sent its first request message (in the current round of request messages). Variable rm

stores the number of request messages that sensor u still needs to send (in the current round of request messages). Variable $rset$ is the multiset of all the replies that sensor u needs to include in its next request-reply message. Variable *done* is a boolean variable whose value is true when and only when the current computation of the strong neighbors of sensor u is completed.

Initially, the value of $nghs$ is the empty set, the value of every element in variable $rcvd$ is 0, the value of variable rm is 0, the value of variable $rset$ is the empty set, the value of variable *done* is true, and the value of implicit variable timer.u is any value in the range 1 .. x.

Each sensor u has two actions that are specified in Fig 3.1.

Sensor u executes its first action when the value of its timer.u becomes zero. The execution of this action starts by checking the value of rm. On one hand, if the value of rm is 0, then u recognizes that it does not need to send a request message, but it needs to send a reply message in case $rset$ is non-empty. Thus, the sent message is of the form `(u,0,rset)`. Also if the value of *done* is true, then sensor u chooses arbitrarily whether it starts to compute its strong neighbors or not. If the value of *done* is false, sensor u invokes a procedure named `COMPNGH` that computes the strong neighbors of sensor u from array $rcvd$ and adds them to the set $nghs$. (In `COMPNGH`, a sensor v is computed to be a strong neighbor of u if $rcvd[v] \geq 2$.) On the other hand, if the value of rm is larger than 0, then u recognizes that it needs to send a request-reply message of the form `(u,1,rset)`.

```
sensor u      // sensor u where 0=< u < n

begin
   timeout-expires ->
       if rm=0 -> if rset != {} -> send (u,0,rset); rset := {}
                  [] rset = {}  -> skip
                  fi;
                  if done  -> skip               // no new round
                  [] done  -> nghs := {};        // start new round
                              rcvd := 0;
                              rm := 3;
                              done := false
                  [] !done -> COMPNGH(in rcvd, out nghs);
                              rcvd := 0;
                              done := true
                  fi; timeout-after random(1,x)
       [] rm>0 -> send (u,1,rset); rset := {};
                  rm := rm-1;
                  if rm>0 -> timeout-after random(1,x)
                  [] rm=0 -> timeout-after random(x+1,x+1)
                  fi
       fi
[] rcv (v,b,s) -> if !done -> rcvd[v] := rcvd[v] + NUM(u,s)
                  [] done  -> skip
                  fi;
                  if b=1 -> rset := rset+{v}
                  [] b=0 -> skip
                  fi
end
```

Figure 3.1: Specification of sensor u in the neighbor computation protocol

Sensor u executes the second action when u receives a `(v,b,s)` message sent by a neighboring sensor v. The execution of this action starts by checking the value of *done*. If the value of *done* is false, then the value of the element $rcvd[v]$ is incremented by $NUM(u, s)$, the number of times u occurs in the multiset s. Then sensor u checks the value of b in the received message. If the value of b is 1, then v is added to the multiset $rset$.

3.3 A Protocol Analysis Example

In this section, we use the analysis methods outlined in Section 3.1 to verify the correctness of the neighbor computation protocol in Section 3.2. Recall that there are three analysis methods: nondeterministic analysis, probabilistic analysis, and collision analysis. We apply each of the three analyses to the neighbor computation protocol in order.

3.3.1 Nondeterministic Analysis

Nondeterministic analysis is used to show that the neighbor computation protocol satisfies some desirable progress property under the two assumptions of idealized message transmission and no message collision. The analysis is carried out from the point of view of a sensor u that needs to compute its strong neighbors.

From the assumption of idealized message transmission, each non-neighbor, weak neighbor or middle neighbor of u cannot receive any message sent by u, or cannot send any message to be received by u. Thus, non-neighbors, weak

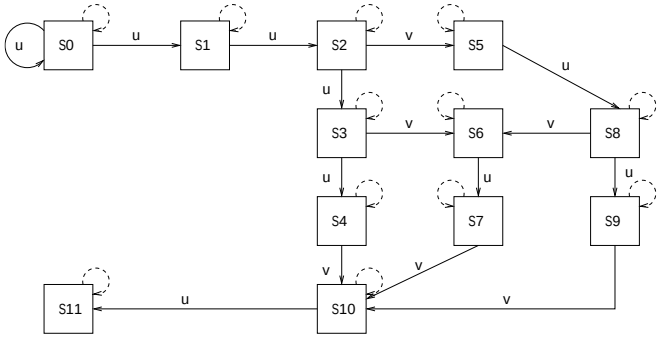


Figure 3.2: State Transition Diagram

S0:			rcvd[v].u=0	$\wedge$	rm.u=0	$\wedge$	done.u=T	$\wedge$ NUM(u, rset.v)=0
S1:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=0	$\wedge$	rm.u=3	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=0
S2:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=0	$\wedge$	rm.u=2	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=1
S3:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=0	$\wedge$	rm.u=1	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=2
S4:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=0	$\wedge$	rm.u=0	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=3
S5:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=1	$\wedge$	rm.u=2	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=0
S6:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=2	$\wedge$	rm.u=1	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=0
S7:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=2	$\wedge$	rm.u=0	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=1
S8:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=1	$\wedge$	rm.u=1	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=1
S9:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=1	$\wedge$	rm.u=0	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=2
S10:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=3	$\wedge$	rm.u=0	$\wedge$	done.u=F	$\wedge$ NUM(u, rset.v)=0
S11:	nghs.u$\ni$ v	$\wedge$	rcvd[v].u=0	$\wedge$	rm.u=0	$\wedge$	done.u=T	$\wedge$ NUM(u, rset.v)=0

Figure 3.3: Specifying the states in the state transition diagram in Fig. 3.2

neighbors and middle neighbors of u have no effect on the computation carried out by u to identify its strong neighbors.

It remains to analyze the interaction between sensor u and each strong neighbor v of u. Fig. 3.2 shows the state transition diagram that describes the interaction between sensor u and its strong neighbor v. Each node in this diagram represents a state of the two sensors u and v. Each dashed edge represents the passing of real-time by one time unit. Each solid edge labeled u represents the execution of the timeout action in sensor u and the execution of

the corresponding receiving action, if any, in sensor v. Each solid edge labeled v represents the execution of the timeout action in sensor v and the execution of the corresponding receiving action, if any, in sensor u.

Each of the states S0 through S11 in the state transition diagram is specified by a predicate in Fig. 3.3. Note that $rcvd[v].u$ is the value of element $rvcd[v]$ in array $rcvd$ in sensor u, $rm.u$ is the value of variable rm in sensor u, $done.u$ is the value of variable $done$ in sensor u, and $NUM(u, rset.v)$ returns the number of times u occurs in the multiset $rset$ in sensor v.

From the state transition diagram, we conclude that the interaction between sensors u and v satisfies the following progress property.

State S1 eventually leads to state S11.

Therefore, the protocol is correct under the two assumptions of idealized message transmission and no message collision.

3.3.2 Probabilistic Analysis

Probabilistic analysis is used to show that the neighbor computing protocol satisfies some desirable progress property, with a high probability, under the relaxation of the first assumption of idealized message transmission.

Under the assumption of idealized message transmission, the middle neighbors and weak neighbors of a sensor u play no role in u's computation of its strong neighbors. When this assumption is relaxed, this is no longer true, and a more refined analysis is in order.

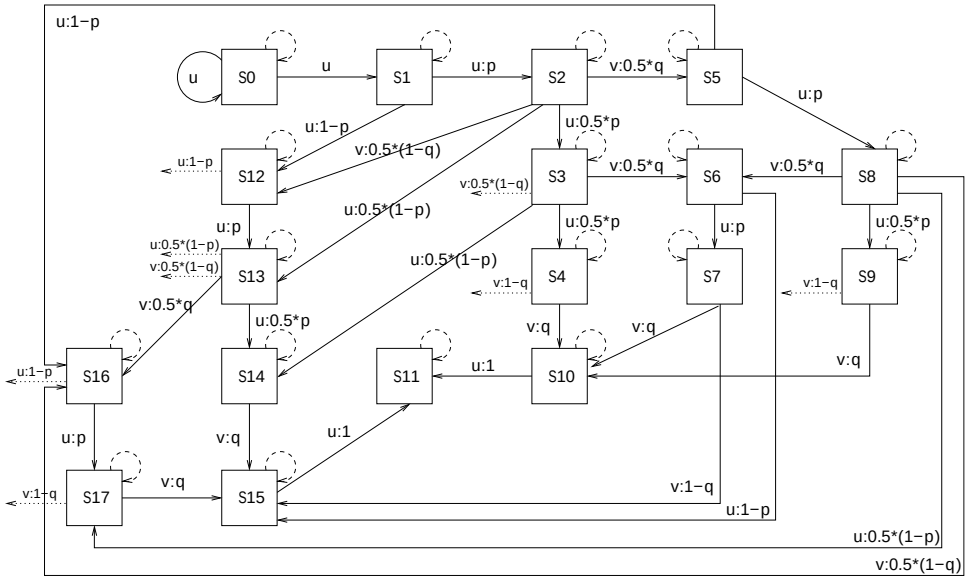


Figure 3.4: Probabilistic State Transition Diagram

Let u and v are distinct sensors in a network. If there are no edges or if there is exactly one edge between u and v in the network topology, then v has no effect on u's computation of its strong neighbors. Otherwise, let there be two edges between u and v in the network topology. Moreover, let p be the probability label of edge (u,v) and q be the probability label of edge (v,u), where $p,q > 0$. In this case, the probability that u identifies v as one of its strong neighbors is the probability that u receives at least two reply messages sent by v, which depends on the probability labels of edges (u,v) and (v,u), p and q.

Fig. 3.4 shows a part of the probabilistic state transition diagram that describes the interaction between sensor u and sensor v, where sensor u receives two or more reply messages sent by sensor v. Each solid edge labeled $u : prob$

S12:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=0	$\wedge$	rm.u=2	$\wedge$	done.u=F	$\wedge$	NUM(u, rset.v)=0
S13:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=0	$\wedge$	rm.u=1	$\wedge$	done.u=F	$\wedge$	NUM(u, rset.v)=1
S14:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=0	$\wedge$	rm.u=0	$\wedge$	done.u=F	$\wedge$	NUM(u, rset.v)=2
S15:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=2	$\wedge$	rm.u=0	$\wedge$	done.u=F	$\wedge$	NUM(u, rset.v)=0
S16:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=1	$\wedge$	rm.u=1	$\wedge$	done.u=F	$\wedge$	NUM(u, rset.v)=0
S17:	nghs.u=$\emptyset$	$\wedge$	rcvd[v].u=1	$\wedge$	rm.u=0	$\wedge$	done.u=F	$\wedge$	NUM(u, rset.v)=1

Figure 3.5: Specifying the states in the state transition diagram in Fig. 3.4

represents the execution of the timeout action in sensor u and the execution of the corresponding receiving action, if any, in sensor v, and this transition occurs with probability *prob*. Similarly each solid edge labeled v : *prob* represents the execution of the timeout action in sensor v and the execution of the corresponding receiving action, if any, in sensor u, and this transition occurs with probability *prob*. Each dotted edge represents a transition yielding to a state, where sensor u receives less than two reply messages sent by sensor v.

Each of the state S12 through S17 in the probabilistic state transition diagram is specified by a predicate in Fig. 3.5. The states of S0 through S11 are the same as in Fig. 3.3.

In the probabilistic state transition diagram, sensor u identifies v as one of its strong neighbor if state S1 eventually leads to state S11. Thus, we need to compute the probability that state S1 eventually leads to state S11 through all 18 different paths from S1 to S11.

The probability to reach S11 from S1 through a path $(s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_k)$, where s_i is a state in the diagram, $s_1 = S1$ and $s_k = S11$, is computed as $\prod_{i=1}^{k-1} e_{i,i+1}$ where $e_{i,i+1}$ is the probability to transit from s_i to

s_{i+1}. For example, the probability to reach S11 from S1 through the path (S1, S12, S13, S14, S15, S11) is $(1-p)*p*(0.5*p)*q$.

Recall that if v is a strong neighbor of u, then both p and q equal 0.95, if v is a middle neighbor of u, then one of the two probabilities is 0.95 and the other is 0.5, and if v is a weak neighbor of u, then both p and q equal 0.5. Substituting these values of p and q in the diagram, we obtain the results in Table 3.1. (Note that the result for a middle neighbor is the average of the two cases of $p, q = 0.95, 0.5$ and $p, q = 0.5, 0.95$.)

Table 3.1: Probability that sensor u identifies sensor v as a strong neighbor in probabilistic analysis

v	Strong neighbor	Middle neighbor	Weak neighbor	Non-neighbor
	0.949	0.472	0.195	0

In the above diagram, some states such as S2, S3, S8, and S13 lead to next states either by the execution of the timeout action in u or by the execution of the timeout action in v. Note that the above analysis is based on the assumption that in such states, the protocol transits to a next state by the execution of the timeout action in u with probability 0.5 and by the execution of the timeout action in v with probability 0.5. However, this assumption is not accurate. The transitions to these states, S2, S3, S8, and S13, are caused by the execution of the timeout action in u, and so the probability of transition from each of these states to a next state by the execution of the timeout action in v may be higher than that by the execution of the timeout action in

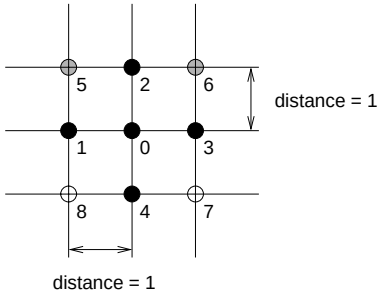


Figure 3.6: The simulated network

u. Thus, in the above analysis, the probabilities to reach S11 through some paths (for example, path (S1,S2,S5,S8,S6,S7,S10,S11)) are computed slightly lower than those that can be obtained from observing an execution of the protocol. However, the probabilities to reach S11 through other paths (for example, path (S1,S2,S3,S4,S10,S11)) are computed slightly higher than those that can be obtained from observing an execution of the protocol.

3.3.3 Collision Analysis

The above probabilistic analysis is based on the assumption of no message collision. In order to take into account the effects of message collision on the execution of a sensor network, one better simulates the network execution and allows message collisions to occur in the simulation.

Consider the sensor network shown in Fig. 3.6. This network has nine sensors: sensor 0 through sensor 8. We assume that only sensor 0 in this network needs to periodically identify its strong neighbors. We also assume that sensor 0 has four strong neighbors, namely sensors 1 through 4, two

middle neighbors, namely sensors 5 and 6, and two weak neighbors, namely sensors 7 and 8.

The "neighboring relation" over the sensors 1 through 8 is determined by the following three rules.

i. If the distance between two sensors is at most 1, the edge is labeled with probability 0.95 (i.e. a strong edge).

ii. If the distance between two sensors is larger than 1 and less than 2, the edge is labeled with probability 0.5 (i.e. a weak edge).

iii. If the distance between two sensors is at least 2, there is no edge between the two sensors.

In order to run the simulation of the neighbor computation protocol, we need to choose the value x. There are two contradictory concerns that can affect our choice of x. If x is large, say 1024, the probability of message collision becomes small, and consequently the probability of correctly identifying a strong neighbor, as measured from the simulation, becomes close to the same probability, as estimated from the probabilistic analysis. On the other hand, if x is large, the average execution time of the protocol, which is around $2 * x + 1$ time units, becomes large.

To choose an appropriate value for x, we ran the simulation three times using different values of x: 64, 128, and 256. In each simulation run, we measured the probability of correctly identifying a strong neighbor. Each of the

simulation results is the average of 1000 runs. At the end, we chose the smallest value of x whose simulation run yielded the probability of correctly identifying a strong neighbor to become around 0.9. From Table 3.2, we conclude that the value of x is 128.

Table 3.2: Probability that sensor 0 identifies sensor v as a strong neighbor in collision analysis

v	1	2	3	4	Avg	5	6	Avg	7	8	Avg
x=64	0.81	0.835	0.851	0.826	0.831	0.413	0.376	0.395	0.132	0.153	0.143
x=128	0.888	0.877	0.897	0.898	0.89	0.435	0.404	0.42	0.157	0.157	0.157
x=256	0.918	0.94	0.935	0.94	0.933	0.468	0.454	0.461	0.169	0.19	0.18

From the simulation run where $x = 128$, we get the following results. For a strong neighbor v, sensor 0 identifies v as a strong neighbor with probability 0.89. For a middle neighbor v, sensor 0 identifies v as a strong neighbor with probability 0.42. For a weak neighbor v, sensor 0 identifies v as a strong neighbor with probability 0.157. (Note that by choosing $x = 128$, the average execution time of the protocol is around 26 seconds, under the assumption that each time unit is 100 milliseconds.)

Table 3.3: Probability that sensor u identifies sensor v as a strong neighbor

Type	Strong ngh	Middle ngh	Weak ngh	Non-ngh
Nondeterministic analysis	1	0	0	0
Probabilistic analysis	0.949	0.472	0.195	0
Collision Analysis	0.89	0.42	0.157	0

As a summary, Table 3.3 shows the probability that a sensor u identifies

another sensor v as a strong neighbor in each of the analysis methods. As we relax the two assumptions of idealized message transmission and no message collision one by one, the probability that u correctly identifies a strong neighbor v is decreased. Moreover, middle neighbors and weak neighbors of u affect u's computation of its strong neighbors.

Chapter 4

Sentries and Sleepers in Sensor Networks

One of the challenging problems in designing sensor networks is to lengthen the lifetime of sensor batteries. One approach to solve this problem is to exploit the idea that in some densely deployed networks, a fraction of the sensors can go to sleep for predefined time periods, while the remaining sensors stay awake and perform the assigned tasks in the network. The sleeping sensors save their energy and lengthen the lifetime of their batteries, without significantly degrading the performance of the applications running on the sensor network. Examples of this approach can be found in [16], [15], [69], [38], [70], [72], [59], [9].

We generalize this idea to be applicable to any, possibly sparsely populated, sensor network: replace each sensor in the network by a group of n sensors, for some $n \geq 2$. The group of n sensors are deployed in a location where a single sensor would have been deployed in the sparse network. This group of n sensors act as one sensor as follows. For a time period, only one sensor in the group, called *sentry*, stays awake and performs all the tasks assigned to the group, while the remaining sensors, called *sleepers*, go to sleep

to save their batteries. At the beginning of the next time period, the sleepers wake up, and all the sensors in the group elect a new sentry for the next time period, and the cycle repeats.

Note that the sensors in a group are identical in every way so that each of them can behave in exactly the same manner in performing the assigned tasks, when this sensor is elected a sentry of the group. This implies that no sensor has an identifier that distinguishes it from other sensors in its group. Rather, every sensor in a group has the same group identifier.

The identifiers of two sensor groups in the same network, however, are distinguishable so that when a sensor receives a message, the sensor can determine whether this message was sent from a sensor in its own group or it was sent from a sensor in a different nearby group. Note that a sensor in a group needs to exchange messages with other sensors in its group in order to elect a new sentry at the beginning of each time period. A sensor also needs to exchange messages with sensors in adjacent groups in order to perform the assigned tasks, when this sensor is elected a sentry of its group.

An alternative approach to lengthen the lifetime of sensor batteries in a sparsely populated network is to provide each sensor with a large battery whose lifetime is n times the lifetime of a regular battery. However, this alternative approach is less reliable than our approach (where each sensor in the sparsely populated network is replaced by a group of n sensors) as follows. If a sensor fails in a network, then the network can compensate for the failed sensor provided the network is designed using our approach rather than the

alternative approach.

The protocol used by a group of n sensors to elect a new sentry at the beginning of each time period is called a *sentry-sleeper protocol*. The goals of a sentry-sleeper protocol are two-fold:

i. Ensure that at each instant not all the sensors in a sensor group are sleeping. Thus, at each instant at least one sensor in the group is awake and so can perform the tasks assigned to the group.

ii. Reduce the time periods where two or more sensors in a sensor group are awake in order to reduce the wasteful use of sensor batteries. (Note that if two or more sensors in a sensor group are awake during a time period, then each of them performs the same tasks assigned to the group during that period.)

When some fault occurs in a network, a sentry-sleeper protocol may reach an illegitimate state that does not satisfy the above goals. Thus, even starting from any illegitimate state, a sentry-sleeper protocol should be able to converge to a legitimate state where at least one sensor in the group is awake, and at most one sensor in the group is a sentry.

Other sentry-sleeper protocols are reported in [15], [9], [16], [70], [38]. The main assumption in these papers is that the sensors in a "sensor group" have distinguishable identities; i.e. they have different physical locations, different connectivity, different message traffic, or different identifiers. Thus, the

sensors in the network decide which one stays awake among their neighboring sensors based on these different identities, so that they can not only save their batteries but also provide some level of the performance of the applications running on the network. Unlike these protocols, our protocol for electing a sentry at the beginning of each time period is based on the assumption that the sensors in a group are perfectly identical; i.e. they have identical locations, connectivity, traffic, and identifiers. This feature makes our protocol scalable and resilient against any attack by an adversary sensor in the group that may lie about its own identity (i.e. lie about their locations, connectivity, ...) to be elected a sentry over and over, and keep the legitimate sensors asleep for a long time.

4.1 Sensor States and Transitions

Before we can explain the main features of our sentry-sleeper protocol, we need to explain, in this section, the different states of a sensor and the transitions between them.

Every sensor in a sensor group can be in any one of two states: an idling state or a sleeping state. In the idling state, the sensor does nothing but wait until either its timeout expires (in which case the sensor executes a timeout action), or it receives a message (in which case the sensor executes a receiving action). An action, whether a timeout action or a receiving action, of a sensor consists of a number of statements that update the local variables of

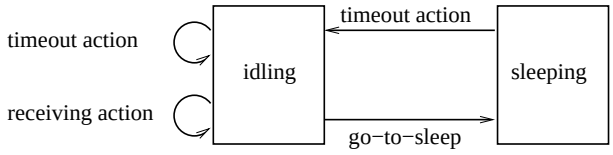


Figure 4.1: Two states of a sensor

the sensor, send at most a message, or set the timeout of the sensor to expire at a later time.

Also the sensor can execute the special statement "go-to-sleep" at the end of an action. If a sensor executes this statement, the sensor changes its state from idling to sleeping. In the sleeping state, the sensor does nothing but wait until its timeout expires, then it executes a timeout action and changes its state from sleeping to idling. Figure 4.1 shows the two states of a sensor and the different transitions between them.

There are two main differences between the idling state and the sleeping state of a sensor. First, in the idling state, the sensor can receive messages that are sent by other sensors and execute corresponding receiving actions, whereas in the sleeping state, the sensor cannot do so since it turns off its radio as well as its processor and sensing devices to save energy during its sleep. Second, the consumed energy when the sensor is in the idling state is much larger than the consumed energy when the sensor is in the sleeping state (as discussed in [51] and [71]). Therefore, for the sensor to save its energy as much as possible, it should stay in its sleeping state as long as possible.

4.2 The Sentry-Sleeper Protocol

The goal of our sentry-sleeper protocol is to make a group of n sensors act as a single sensor whose lifetime is $N * F$ time units, where F is the lifetime of a regular sensor, and $1 < N < n$. The n sensors in the sensor group constitute a sensor network whose topology is fully-connected, i.e. there are two opposite-direction edges between every two nodes in the topology.

During a time period, called a *turn*, $(n-1)$ sensors of the sensor group are in their sleeping states and the remaining sensor is in its idling state. In a turn, each of the sleeping sensors is called a *sleeper*, and the awake sensor is called a *sentry*. At the end of a turn, the sleepers wake up and all sensors in the group elect a new sentry for the next turn. This cycle of a turn followed by an election of a new sentry is repeated over and over until the batteries of all sensors in the group are exhausted.

At the end of a turn, the sleepers wake up, and they along with the sentry collaborate to elect a new sentry for the next turn as follows. Each sensor in the group computes a random period, called a *resolution period*, whose length is chosen uniformly from the range 1 .. $2^{*}ravg-1$, where $ravg$ is the average length (measured in time units) of the resolution period. Then, each sensor sets its timeout to expire after its resolution period. The sensor that chooses the smallest resolution period in the group times-out first, and this sensor elects itself as the new sentry and starts the new turn by sending a message of the form:

`sleep(`gid`, `rt`)`

where gid is the identifier of the sensor group and rt is the remaining time in the current turn. Initially, the remaining time in the current turn is assigned the length of a turn, which is tl time units.

When a sensor u in the sensor group receives a sleep(gid, rt) message, sensor u recognizes that a new sentry is elected for the current turn and decides to sleep for rt time units. Thus, it sets its timeout to expire after rt time units, then goes to sleep. The range of rt in the received sleep message is 1 .. tl. Thus, the shortest sleeping period is 1 time unit, and the longest sleeping period is tl time units.

After the elected sentry sends the first sleep(gid, rt) message, the sentry computes a random period whose length rp is chosen uniformly from the range 1 .. 2*$ravg$−1, and sets its timeout to expire after rp time units. When the sentry times-out, it sends the next sleep(gid, $rt - rp$) message. The sentry keeps on sending sleep messages, until the remaining time in the current turn becomes zero and all the sleepers wake up to elect a new sentry for the next turn.

Notice that the sentry periodically sends a sleep message even when all other sensors in the group are supposedly asleep and cannot receive any messages. This feature is intended to handle the following case. Some sensors in the group may not receive the first sleep(gid, rt) message sent by the sentry at the beginning of a turn. These sensors can receive a later sleep(gid, rt')

message, where $rt' < rt$ and go to sleep for a period of rt' time units in this turn.

In this protocol, two (or more) sensors, say u and v, in the group can select identical resolution periods and so they send their sleep messages at the same time. The net effect is that none of the sensors in the group can receive any sleep messages, since the two messages collide with one another. Only sensors u and v consider themselves as sentries, and the other sensors in the group do not recognize that a new sentry has been elected for the current turn. However, our protocol ensures that one, only one, sensor in the group eventually sends a sleep message at some instant t and makes all other sensors go to sleep at t.

A formal specification for a sensor u in the group is given in Fig. 4.2.

It is important to note that in this protocol, the sensors in a group compete to become a sentry purely based on randomization without resorting to any difference in their identities that may give an advantage to some sensors over others in the group. In a turn, each sensor in the group has the same probability to become a sentry. Thus, each sensor can expect to become a sentry once every n turns or so. A sensor u who fails to become a sentry for a relatively long period, say for $3 * n$ or $5 * n$ turns, should suspect that some sensors in the group are not following the protocol. In this case, sensor u may decide to stay awake (and perform the assigned tasks to the group) and refuse to go to sleep.

```
sensor u
const gid     : integer,     {group id of sensor u}
      tl      : integer,     {length of a turn}
      ravg    : integer      {avg length of random period}
var   sentry  : boolean,     {Is u sentry?}
      awake   : boolean,     {Is u awake?}
      rp      : 1..2*ravg-1, {length of random period}
      rt      : 0..tl,       {remaining time in current turn}
      g       : integer,     {group id in received message}
      t       : 1..tl        {remaining time in received message}
begin
    timeout-expires -> if !awake ->            awake := true;
                                               sentry := false;
                                               rp := random;
                                               timeout-after rp
                       [] awake and !sentry -> sentry := true;
                                               rt := tl;
                                               send sleep(gid, rt);
                                               rp := random;
                                               rp := min(rp, rt);
                                               rt := rt-rp;
                                               timeout-after rp
                       [] awake and sentry ->
                                    if rt>0 -> send sleep(gid, rt);
                                               rp := random;
                                               rp := min(rp, rt);
                                               rt := rt-rp;
                                               timeout-after rp
                                    [] rt=0 -> sentry := false;
                                               rp := random;
                                               timeout-after rp
                                    fi
                       fi
[]  rcv sleep(g, t) -> if gid=g  -> sentry := false;
                                    awake := false;
                                    timeout-after t;
                                    go-to-sleep
                       [] gid!=g -> skip
                       fi
end
```

Figure 4.2: Specification of sensor u in the sentry-sleeper protocol

4.3 Stabilization of the Protocol

In this section, we sketch a proof that our sentry-sleeper protocol is self-stabilizing. A *state* of this protocol is defined by a value for each variable and each implicit variable timer.u for each sensor u in a group.

We assume that every state (whether legitimate or illegitimate) of the protocol satisfies the following four conditions.

1. For every sensor u, the value of variable timer.u is at most tl time units. (Note that this assumption is maintained by the execution of the protocol.)

2. For every sleeping sensor u, the value of its *awake* variable is false. (Note that this assumption is maintained by the execution of the protocol.)

3. For every awake sensor u, the value of its timer.u is distinct from the value of timer.v for any other awake sensor v. (Note that this assumption is probabilistically maintained by choosing the value $ravg$ to be large relative to the number of sensors in the group.)

4. For every awake sensor u, if sensor u sends a message at instant t, then an awake sensor v receives a copy of the message at t, provided that sensor v does not send a message at t, and no other awake sensor except u sends a message at t.

In our sentry-sleeper protocol, a *legitimate* state is defined as a state that satisfies the following *invariant*:

At least one sensor in the group is awake, and

at most one sensor in the group is a sentry.

Therefore, in a legitimate state, the number of sleepers is in the range 0 .. $n-1$, and the number of sentries is in the range 0 .. 1.

The protocol is *self-stabilizing* iff it satisfies the following two conditions [5].

i. *Closure*: Starting from any legitimate state, the execution of any action in any sensor in the protocol yields a legitimate state.

ii. *Convergence*: Starting from any illegitimate state, the protocol is guaranteed to reach a legitimate state.

First, we show that starting from any legitimate state, the execution of any action in any sensor in the protocol yields a legitimate state. The protocol has two cases to consider. In the first case, the executed action is a timeout action in a sensor u in the group. In this case, there are three possibilities to consider when the timeout action is executed.

i. The value of *awake* in u is false: In this case, u concludes that u wakes up from sleeping (by the assumption 2), and makes the value of *awake* true. Thus, u becomes awake, and so the invariant holds.

ii. The value of *awake* in u is true and the value of *sentry* in u is false: In this case, u elects itself as the new sentry and makes other sensors in

the group sleep by sending a sleep message. Note that no other awake sensor can execute this timeout action that causes the sensor to send a sleep message at the same time (by the assumption 3). Thus, u is awake and becomes the only sentry in the group, and so the invariant holds.

iii. The value of *awake* in u is true and the value of *sentry* in u is true: In this case, there are two cases to consider depending on the remaining time in the current turn. First, if the remaining time is bigger than zero, u sends another sleep message. Thus, u is still awake and is still the only sentry in the group. Second, if the remaining time is zero, u recognizes that the current turn is finished, and withdraws from a sentry by making its value of *sentry* false. Thus, u is still awake, but is not a sentry any more. In both cases, the invariant holds.

In the second case, the executed action is a receiving action in a sensor u in the group. In this case, there are two possibilities to consider when the receiving action is executed.

i. When u receives a sleep message from another sensor v in the same group: In this case, u recognizes that sensor v is elected a sentry for the current turn, so u goes to sleep for the specified sleeping period in the received message. Thus, sensor v is awake and is the only sentry in the group, and so the invariant holds.

ii. When u receives a sleep message from a sensor in a different group: In

this case, u ignores the message and does nothing. Thus, the invariant holds.

Hence, starting from any legitimate state, the execution of a timeout action or a receiving action in any sensor in the group yields to a legitimate state.

Next, we show that starting from any illegitimate state, our protocol is guaranteed to reach a legitimate state within finite executions of actions in the group. There are two states that violate the invariant as follows:

i. A state where all sensors in the group are sleeping: In this case, a sensor u in the group is guaranteed to execute its timeout action within tl time units (by the assumption 1). By executing the timeout action of u, u becomes awake. Thus, at least one sensor in the group will wake up within tl time units, and then only one of the awake sensors will eventually become a sentry.

ii. A state where two or more sentries exist in the group: In this case, only one sentry whose timer value is the smallest, say sensor u, times-out first and then executes its timeout action to send a sleep message at some instant t (by the assumption 3). The other sentries receive the sleep message from u and go to sleep at t. Thus, all the sentries except u go to sleep within finite executions of actions.

Therefore, starting from any illegitimate state, the execution of a timeout action in some sensor makes the protocol reach a legitimate state within finite executions of actions in the group.

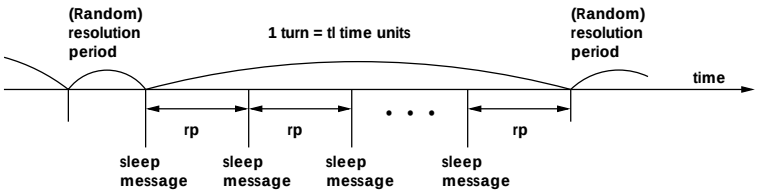


Figure 4.3: A time period during protocol execution

4.4 Protocol Analysis

Our protocol, as described in Section 4.2, makes a group of n sensors act as one sensor whose lifetime is $N * F$ time units, where F is the lifetime of a regular sensor and N is some quantity, called *the effective number of the sensors* in the group. Clearly, we have $1 < N < n$. In this section, we analyze the protocol and estimate the value of N.

Figure 4.3 shows a time period T, consisting of a resolution period followed by one turn of tl time units. Since the average length of a resolution period is $ravg$ time units, we have $T = tl + ravg$ time units. During a turn, a sentry sends a sleep message every random period rp whose average is $ravg$ time units. Therefore, the average number of sleep messages sent by a sentry per turn is $tl/ravg$.

Let E_{slp} and E_{idl} be the amount of energy consumed by a sensor in the sleeping state and in the idling state per time unit respectively, and E_{snd} and E_{rcv} be the amount of energy consumed by a sensor to send a message and to receive a message respectively. There are two possible cases that can occur during the time period T:

i. *Case 1*: The sensors in the group do not execute the protocol, and remain in their idling states. The amount of energy consumed by n sensors in this case, E_{nop} is calculated as follows.

$$E_{nop} = E_{idl} * (tl + ravg) * n$$

ii. *Case 2*: The sensors in the group execute the protocol. The sentry stays in the idling state and sends $tl/ravg$ sleep messages for this time period. Each of the $(n-1)$ sleepers stays in the idling state for a resolution period, receives a sleep message, and sleeps for tl time units. Therefore, the amount of energy consumed by n sensors in this case, E_p is calculated as follows.

$$\begin{aligned} E_p = \quad & E_{idl} * (tl + ravg) + E_{snd} * (tl/ravg) \\ & +(n-1) * (E_{slp} * tl + E_{idl} * ravg + E_{rcv}) \end{aligned}$$

From the above analysis, we can estimate the effective number of the sensors N as follows:

$$N = \frac{E_{nop}}{E_p}$$

We present two figures, Figure 4.4 and 4.5, from the above formula for the four cases $tl = 30 * ravg$, $60 * ravg$, $90 * ravg$ and $120 * ravg$ in time units. In both of the figures, we use the values in Table 4.1 for E_{slp}, E_{idl}, E_{snd} and E_{rcv}. These values are computed using the energy consumption model in [51],

Table 4.1: Energy consumption of a sensor (in energy units)

E_{slp}		0.003	per time unit
E_{idl}		30	per time unit
E_{snd}		24.3	per message
E_{rcv}		9	per message

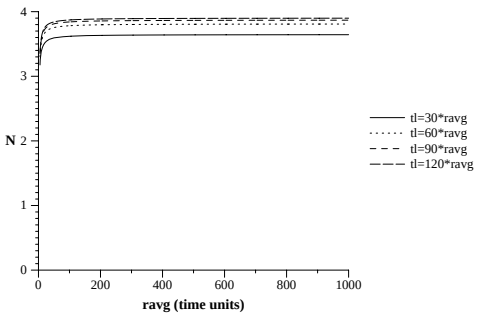


Figure 4.4: N vs. $ravg$ when n=4

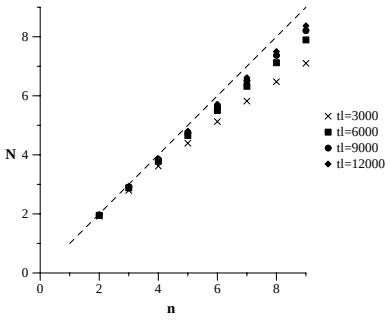


Figure 4.5: N vs. n when $ravg$=100

under the assumption that a time unit is 100 milliseconds, and a time period taken for a sensor to send or receive a message is 30 milliseconds.

Figure 4.4 shows the relationship between the length of $ravg$ and N when n=4 and $5 \leq ravg \leq 1000$ in time units. If $ravg$ is 100 time units or more, then the value of N no longer depends on $ravg$. Similarly, when n=2 and 9, if $ravg$ is 100 time units or more, then the value of N no longer depends on $ravg$.

Figure 4.5 shows the relationship between n and N when $ravg = 100$

time units. From Figure 4.5, one can make two observations. First, tl does not have a strong effect on the value of N, especially when n is small. Second, N is closer to n when n is smaller. During a resolution period, all sensors in the group need to stay awake, and so the total amount of energy consumed by the sensors during this period is increased as n is increased. Thus, our protocol becomes more efficient as n is smaller.

In the real execution of the protocol, the current sentry can run out of the battery and die during its turn. Then there exists a time period where no sensor in the group is awake to perform the tasks assigned to the group. We call this time period a *gap*.

We can estimate the total length of gaps over the lifetime of a group of n sensors from a simple formula. When the sentry dies during its turn, the average time period that no sensor in the group is awake is $tl/2$ (because the minimum time period is zero and the maximum time period is tl). Since $(n-1)$ sensors can die during their sentry turns, the total length of gaps over the lifetime is estimated as follows:

$$\frac{tl}{2} * (n-1)$$

The total length of gaps is relatively much smaller than the lifetime of the group. Note that the total length of gaps is related to the number of sensors in the group, not the lifetime of a regular sensor.

4.5 Simulation Results

We have developed a simulator that can simulate the execution of our sentry-sleeper protocol. This simulator simulates the behavior of a group of n sensors whose topology is fully connected and also allows us to configure the parameters of the protocol such as tl and $ravg$.

For the purpose of simulation, we have adopted the values in Table 4.1 as well as the following values:

- tl= 3000 time units
- $ravg$= 100 time units
- The amount of energy given to each sensor, at the beginning of simulation, is enough to keep that sensor in its idling state for 100000 time units.

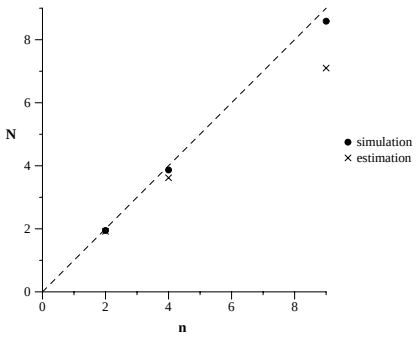


Figure 4.6: The effective number of the sensors

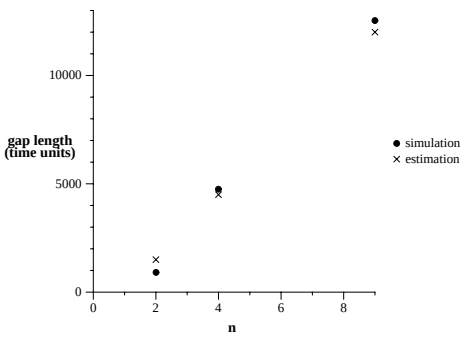


Figure 4.7: The total length of gaps

We ran simulations of this protocol for the three cases $n = 2$, 4 and 9, and plotted the results in Figure 4.6 and 4.7. Figure 4.6 shows the effective

number of the sensors N. Each circle mark represents the average effective number of the sensors over 100 simulations and each X mark represents the estimated effective number of the sensors. The effective number of the sensors in simulation is larger than that in estimation, because in simulation, sensors run out of their batteries and die over time and so the number of sensors in the group decreases over time. As discussed in Section 4.4, the protocol becomes more efficient as n is smaller.

Figure 4.7 shows the total length of gaps over the lifetime of a group of n sensors. Each circle mark represents the average total length of gaps over 100 simulations and each X mark represents the estimated total length of gaps.

From the simulation results, we show that the effective number of the sensors N is close to the number of sensors n in the group. That is, the group of n sensors can lengthen its lifetime around n times the lifetime of a regular sensor by adopting our protocol.

4.6 Related Work

It is suggested in SBPM[38] to divide the sensors in a network into two sets, sentries and sleepers. Sentries stay awake, and provide basic communication services and coarse sensing services, while sleepers go to sleep to save their energy. When the sentries detect events, they can wake up the sleepers for more refined sensing. However, in SBPM, sentries are pre-selected and fixed. Moreover, a central computer decides when sleepers go to sleep. In GAF[70],

all sensors that are equivalent in routing are identified using geographical location information. Then, only one sensor in a group of equivalent sensors stays awake and participates in routing, while the other sensors turn off their radios and go to sleep.

In Span[16] and TMPO[9], each sensor exchanges its neighbor information to compute which sensor joins a connected backbone in a network. Only sensors in the backbone participate in routing, while other sensors can go to sleep to save energy. In ASCENT[15], a sensor in a network keeps track of the number of its active neighbors and message loss rate, and joins a network topology only if the sensor becomes helpful. However, once a sensor enters the active state, it continues to be awake until it dies.

Other approaches to save energy in a sensor network have been proposed in [32], [62], [71], [59], [69], [72]. In LEACH[32], to reduce communication cost, each cluster-head collects data messages from the sensors in the cluster, and then compresses and forwards the messages to a base station. In STEM[59], a sensor in a monitoring state turns off the radio. If the sensor detects an event, it turns on the radio and wakes up other sensors if necessary.

Leader election protocols have been studied for single-hop single-channel radio networks in [52], [40], [41] and [30]. However, in general, these protocols assume that a station sending a message can simultaneously listen [52], [40], [30]. These protocols may not be useful in a sensor network, since generally a sensor cannot listen while the sensor is sending a message as in IEEE 802.11.

Chapter 5

Logical Grid Routing in Sensor Networks

A sensor network usually supports two communication patterns between the sensors in the network: unicast and broadcast. In the unicast pattern, any sensor in the network can send a message whose ultimate destination is the base station. This pattern is used, for example, when a sensor senses an event and needs to report the event to the base station. In the broadcast pattern, the base station can send a message whose ultimate destination is every sensor in the network. This pattern is used, for example, to tune parameters at the different sensors or reset the whole network. A routing tree whose root is the base station can be used to provide both communication patterns.

It is difficult to design routing protocols for sensor networks. This is because these routing protocols need to overcome the special challenges that are posed by sensor networks. First, a sensor network has limited resources. Examples of these resources are the memory available in each sensor, the energy remaining for each sensor, and the communication capacity of the sensor network. Any routing protocol for sensor networks should not consume a large fraction of the network resources. For example, sensors should not store a large

routing table. Also, sensors should not send large routing messages frequently.

Second, sensors in a sensor network may start to execute the routing protocol from an illegitimate state. Also the sensors can be unreliable. For example, some sensors in the network may fail-stop when they run out of their energy or when they are physically damaged. Any routing protocol for sensor networks should be able to stabilize to a legitimate state, starting from any state, and also should be able to recover from the situation where a sizable fraction of the sensors fail-stop.

Third, several previous experimental studies showed the existence of unreliable long links in sensor networks [14, 25, 60, 68, 76]. When sensors in a sensor network build a routing tree, these unreliable links should be avoided, since they result in poor message delivery. Not all long links are unreliable, but sensors cannot easily identify which long links are reliable and which are unreliable, especially when the traffic and the environment are changing dynamically. For this reason, distance vector routing protocols [22] may not work well in sensor networks, since these protocols take advantage of all long links but many of them are unreliable. One approach to avoid unreliable links is to estimate link quality dynamically using beacons [60, 68]. Nevertheless, link quality is highly affected by environment, traffic pattern, and interference [68, 74, 76], and so it is hard to estimate link quality precisely, especially under bursty traffic which is a common traffic pattern in sensor networks [74].

In this chapter, we present a routing protocol, called the logical grid routing (LGR) protocol, that overcomes the challenges of sensor networks.

First, the protocol is simple and so it consumes a small percentage of the network resources. In particular, the protocol requires that every sensor in the network sends only one routing message that consists of $2 - 4$ bytes every 20 seconds (or less frequently), and stores no more than $10 - 15$ bytes of routing information. Second, the routing protocol is stabilizing such that starting from any state, the protocol converges to a state where all the sensors and only the sensors that can be connected to the routing tree are connected to the tree. We show that the convergence time of the protocol is proportional to the diameter of the sensor network. We also show that even if 50% of the sensors in a network fail-stop, 84% of the remaining sensors in the network can still route data messages. Third, we adopt a simple approach to avoid unreliable long links in the routing tree. In this approach, we use off-line experiments to estimate link quality, and use these results to identify reliable links in the routing tree. The experiments in our testbed showed that this simple approach works well in practice. Moreover, the LGR protocol has been used to successfully support reliable delivery of bursty traffic in a large scale sensor network [7].

The rest of the chapter is organized as follows. In Section 5.1, we discuss a logical grid and potential parents of sensors in the logical grid. We present the LGR protocol in Section 5.2 and prove that this protocol is stabilizing in Section 5.3. We present the extended protocol with foster parents, which allows a sensor to have a "foster parent" in the tree when all its potential parents fail-stop in Section 5.4 and prove that this protocol is stabilizing in

Section 5.5. We show the simulation and experimental results of the LGR protocol in Section 5.6. In Section 5.7, we discuss the advantages and limitations of the LGR protocol. We discuss related work in Section 5.8.

5.1 Logical Grid and Potential Parents

In this section, we first describe a logical grid and potential parents of sensors in the logical grid, and present an algorithm to compute potential parents for each sensor. We then discuss one of the methods to build a logical grid.

We consider a sensor network where sensors are deployed at arbitrary physical locations, but they are named as if they form an $M * N$ logical grid. In this network, each sensor is identified by a pair (i,j), called the sensor identifier in the logical grid, where $i = 0..M-1$ and $j = 0..N-1$. The base station in this network is sensor (0,0).

(To keep our presentation simple, we choose the base station to be the grid point (0,0). In practice, it is advantageous to select the base station to be the grid point ($M/2$, $N/2$) at the middle of the logical grid. In this case, the maximum number of hops to be traveled by data messages from any sensor to the base station is minimized. Moreover, the probability that the base station can be separated from the rest of the network is also minimized.)

When a sensor (i,j) has a data item and wishes to send this data item to its final destination, the base station (0,0), sensor (i,j) forwards the data

item to a neighbor (i',j'), where $i \geq i'$ and $j \geq j'$, that is closer towards (0,0) than (i,j). The transmission of the data item from sensor (i,j) to sensor (i',j') is called *one hop*. The "physical" distance between sensors (i,j) and (i',j') should be small enough in order not to create a long link, but should be large enough in order to reduce the number of hops until the data item reaches its final destination.

Roughly, the requirement that sensor (i,j) forwards its data item to a neighbor (i',j') can be stated as follows:

$$(i - i') + (j - j') = H$$

where H is a small positive integer, called the *hop size*. In this case, sensor (i',j') is called a *potential parent* of sensor (i,j) in the routing tree whose root is the base station (0,0). Thus, sensor (i,j) can have up to $H + 1$ potential parents, sensors $(i,\ j - H)$, $(i - 1,\ j - H + 1)$ ... $(i - H + 1,\ j - 1)$, and $(i - H,\ j)$. Note that the above characterization of potential parents may not be valid for sensors on or near the boundary of the grid. We describe below an algorithm to compute potential parents of each sensor including those on or near the boundary of the grid.

It follows from the above discussion that if H is chosen to be two, then each sensor (i,j) in the logical grid has up to three potential parents. For example, referring to a logical grid in Fig. 5.1, the potential parents of sensor (3,3) are sensors (1,3), (2,2), (3,1). However, not every sensor has three potential parents in this case. For example, the base station (0,0) has

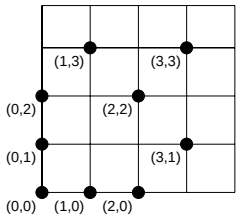


Figure 5.1: A 5*5 logical grid where the hop size is 2

no potential parent. Each of the two sensors (0,1) and (1,0) has only one potential parent, namely the base station (0,0). Also, sensor (0,2) has two potential parents (0,1), (0,0), and sensor (2,0) has two potential parents (0,0), (1,0).

The algorithm in Fig. 5.2 can be used by any sensor (i,j) to compute set P of its potential parents in an $M * N$ logical grid where the hop size is H.

Next, we discuss one of the methods to build a logical grid and compute the value of H. (Similar methods to build a logical grid for the case $H = 1$ can be found in [24, 70].) We divide a network area into grid squares where each size is $d * d$. In this network, a sensor is assigned a logical identifier (i,j), if the sensor is deployed at the grid square (i,j). The sensor acts as if it is deployed at the grid point (i,j), that is the central point of the grid square, of the logical grid. For example, Fig. 5.3 shows a network area that consists of three grid squares. In Fig. 5.3, a dashed square represents each grid square, and solid lines represent the imposed logical grid for the network area. Note that the value of d needs to be selected such that at least one sensor is deployed at each grid square with a high probability.

```
Algorithm

inputs    M,N : integer,           // M*N grid
          H   : integer,           // hop size
          i   : 0..M-1,
          j   : 0..N-1

outputs   set P of potential parents of sensor (i,j)
          in an (M*N) grid whose hop size is H

variables u,v : 0..H

begin
    u := 0;
    do u <= H -> v := H-u;
                  if i-u>=0 and j-v>=0 -> add sensor (i-u, j-v) to set P
                  [] i-u<0  and j-v>=0 -> add sensor (0, j-v) to set P
                  [] i-u>=0 and j-v<0  -> add sensor (i-u, 0) to set P
                  [] i-u<0  and j-v<0  -> skip
                  fi;
                  u := u+1
    od;
    remove sensor (i,j) from set P
end
```

Figure 5.2: Algorithm to compute potential parents

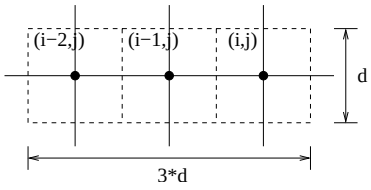


Figure 5.3: Network area

We assume that each sensor knows its physical location (x,y) using GPS or localization services in [36, 61] and the side length of a grid square d. We also assume that the reliable transmission range of sensors is R. Using its physical location and d, each sensor decides its logical identifier (i,j). (In this work, we assume that only one sensor is deployed at each grid square. If two or more sensors are deployed at the same grid square, only one sensor among them can be selected and stay awake to save energy by using an energy saving protocol such as [28, 70].)

In order to establish that the imposed logical grid and the selected H are valid, each sensor should be able to "reliably" exchange messages with each of its potential parents. In other words, R should be larger than the maximum distance between a sensor and its potential parents. Therefore, we need the following condition.

$$R \geq \sqrt{d^2 + ((H+1) * d)^2} \qquad \text{i.e.} \qquad R \geq d * \sqrt{H^2 + 2H + 2}$$

For example, consider the case $H = 2$. In this case, a sensor (i,j) has the maximum distance with its potential parent (i-2, j) (or (i,j-2)). The maximum distance between two sensors deployed at grid squares (i,j) and (i-2,j) is computed as $d * \sqrt{10}$, as shown in Fig. 5.3. Thus, R should be at least $d * \sqrt{10}$.

We perform off-line experiments to find the reliable transmission range R of sensors, and then we select the largest integer H that satisfies the above condition. Note that we need to conservatively measure R so that a sensor

can communicate with each of its potential parents reliably, even under heavy interfering traffic.

From now on, we assume that the imposed logical grid on the sensor network and the selected hop size are valid. We also assume that neighbor relation is symmetric. Therefore, if a sensor u can receive a message sent by another sensor v, sensor v can receive a message sent by sensor u.

5.2 The Logical Grid Routing Protocol

The purpose of the LGR protocol is to build and maintain a routing tree whose root is the base station (0,0). Each sensor (i,j) chooses one of its potential parents to be its parent in this tree.

Initially, only the base station (0,0) is in the routing tree. The base station periodically sends a message of the form, connected(0,0), every random period whose length is chosen uniformly from the range $rmin$ .. $rmax$, where the average length of the range $rmin$ .. $rmax$ is T time units. (This random period is to reduce the probability of connected message collision.)

When a sensor (i,j), that is currently not connected to the tree, receives a connected(i',j') message and checks that sensor (i',j') is one of its potential parents, sensor (i,j) becomes *connected* to the tree and makes sensor (i',j') its parent. From this point on, sensor (i,j) sends a connected(i,j) message periodically every random period whose length is chosen uniformly from the range $rmin$ .. $rmax$.

When a sensor (i,j), that is currently connected to the tree and whose parent is sensor (i',j'), receives a connected(i'',j'') message and checks that sensor (i'',j'') is one of its potential parents, sensor (i,j) remains connected to the tree, but changes its parent in the tree from sensor (i',j') to sensor (i'',j''). In this case, sensor (i,j) continues to send a connected(i,j) message periodically every random period whose length is chosen uniformly from the range $rmin$ .. $rmax$.

When a sensor (i,j), that is currently connected to the tree and whose parent is sensor (i',j'), does not receive any connected(i'',j'') message from any of its potential parents for a time period of $tmax * T$ time units, sensor (i,j) concludes that it is no longer connected to the tree and stops sending connected(i,j) messages. (We discuss how to choose a value of $tmax$ below.) Later when sensor (i,j) receives a connected(i'',j'') message and checks that sensor (i'',j'') is one of its potential parents, sensor (i,j) becomes connected again to the tree and makes sensor (i'',j'') its parent in the tree.

A specification for the base station is given in Fig. 5.4.

We mentioned earlier that when a sensor does not receive a connected message from any of its potential parents for the time period of $tmax * T$ time units, the sensor recognizes that it is no longer connected to the routing tree. This feature is implemented by providing each sensor (i,j), where $i \neq 0$ or $j \neq 0$, with a variable trc whose value is in the range 0 .. $tmax$. When sensor (i,j) receives a connected message from any of its potential parents, trc is assigned the value $tmax$. Every time sensor (i,j) times-out to send a

```
sensor (0,0)                  // base station

const rmin,rmax : integer     // min and max interval whose avg is T
var   r         : rmin..rmax  // random interval to send next connected msg

begin
   time-out expires -> send connected(0,0);
                       r := rand; time-out after r
end
```

Figure 5.4: Specification of sensor (0,0) in the LGR protocol

connected message, the value of variable trc is decremented by one. When the value of variable trc becomes zero, sensor (i,j) recognizes that it is no longer connected to the routing tree.

During the execution of the protocol, a sensor u may consecutively choose small random periods for some number of times, and some of connected messages sent by the potential parents of sensor u may be lost. Nonetheless, we assume that sensor u is truly disconnected from the routing tree if the value of trc in u becomes zero. In order to establish that our assumption is valid, we need to assign $tmax$ a reasonably large value that ensures the following with a high probability: if any potential parent v of a sensor u sends a connected message periodically, then sensor u is guaranteed to receive a connected message sent by v before the value of trc in u becomes zero. (Note that this value of $tmax$ can be estimated based on the values of $rmin$ and $rmax$, and the probability of message loss in the network.) From now on, we

assume that a reasonable value that makes our assumption valid is assigned to $tmax$.

A specification for sensor (i,j), where $i \neq 0$ or $j \neq 0$, is given in Fig. 5.5.

```
sensor (i,j)           // a sensor (i,j) in an M*N grid where i=!0 or j!=0

const P         : set of potential parents of sensor (i,j),
      rmin,rmax : integer,           // min and max interval whose avg is T
      tmax      : integer            // max time to be connected

var   pid       : an element from P, // parent identifier
      trc       : 0..tmax,           // time to remain connected
      r         : rmin..rmax,        // random interval to send connected msg
      x         : 0..M-1,
      y         : 0..N-1

begin
   rcv connected(x,y) ->  if (x,y) in P    -> pid := (x,y); // choose parent
                                              trc := tmax
                          [] !((x,y) in P) -> skip
                          fi

[] time-out expires ->    trc := max(trc-1,0);
                          if trc>0 -> send connected(i,j)
                          [] trc=0 -> skip                  // lose parent
                          fi; r := rand; time-out after r
end
```

Figure 5.5: Specification of sensor (i,j) in the LGR protocol

Notice that in every state of the protocol, the value of timer.u for every sensor u in the network is at most $rmax$. (This is maintained by the execution of the protocol.)

In the LGR protocol, the parent of a sensor at any time is the last potential parent from which this sensor received a connected message. In other words, a sensor keeps changing its parent whenever it receives a connected message from another potential parent. This feature provides two nice properties: load balancing and fast fault recovery. First, load balancing is achieved, since the data messages, that are generated at the same sensor, are likely to follow different routes to the base station. Second, fast fault recovery is achieved when the current parent of a sensor fail-stops. In this case, the sensor replaces this parent as soon as it receives a connected message from another potential parent. (Note that this feature may cause the arrival of data messages at the base station out of order. However, this is not a problem in sensor networks since most data messages are tagged with the real-time of when they were generated.)

Next, we specify how this protocol is used to route data messages to the base station (0,0). When a sensor (i,j), other than the base station, has a data message to route to the base station, sensor (i,j) first checks whether or not it is connected to the routing tree. If the value of variable trc in sensor (i,j) is more than zero, sensor (i,j) concludes that it is connected to the tree. In this case, sensor (i,j) sends the message after attaching its parent identifier pid to the message. Otherwise, sensor (i,j) recognizes that it is not connected to the tree and drops the message.

When the base station (0,0) receives any data(i,j) message, the base station accepts the data message even if (i,j) is not (0,0). This is because the

ultimate destination of all data messages is the base station. Note that the same data message may be received by the base station more than once, since the data message is still forwarded along the routing tree until it reaches the base station. Thus, the snooping feature of the base station can only increase the probability that every data message is received (at least once) by the base station.

5.3 Stabilization of the Protocol

In this section, we sketch a proof that the protocol in Section 5.2 is self-stabilizing. This proof is based on the following four assumptions.

A1) Each sensor in the protocol is either up or down. Up sensors execute the protocol as desired, while down sensors just fail-stop. Moreover, whether a sensor is up or down does not change along any execution of the protocol.

A2) For every two distinct up sensors u and v, if u is a neighbor of v, or if there exists a third sensor w that is a neighbor for both u and v, then $timer.u$ and $timer.v$ have distinct values. (Note that this assumption is probabilistically maintained by choosing the difference $rmax - rmin$ to be large relative to the number of neighbors of a sensor.)

A3) For every up sensor u, if the value of trc in sensor u is 1, then sensor u chooses a value for $timer.u$ such that the value of $timer.u$ is larger than

the value of $timer.v$ for any up potential parent v of sensor u.

A4) For every up sensor u, if the value of trc in sensor u is 0 or 1, and any potential parent of sensor u sends a connected message, then sensor u receives the connected message.

(Note that assumptions A3 and A4 ensure that if any potential parent v of a sensor u sends a connected message periodically, then sensor u is guaranteed to receive a connected message sent by v before the value of trc in u becomes 0.)

In the protocol, a *state* of a sensor u is defined by the value of variable trc and the value of variable $timer.u$ in sensor u, and the state of each up potential parent v of sensor u. Sensor u is in one of the following two states: (We use the notation <var> $.u$ to denote the value of variable <var> in sensor u.)

i. Sensor u is in a *c-state* iff u is the base station, or at this state
($trc.u > 1$ and there exists an up potential parent v of u where v is in a c-state) or
($trc.u = 1$ and there exists an up potential parent v of u where v is in a c-state and $timer.u > timer.v$)

ii. If sensor u is not in a c-state, then sensor u is in a *d-state*.

(During the execution of the protocol, the value of $timer.u$, as well as other variables, in sensor u can be corrupted, but we assume that the protocol is

implemented such that in every state, the value of $timer.u$ is at most $rmax$.)

A *configuration* of the protocol is defined by the state for each up sensor in the protocol.

Regardless of the initial configuration of the protocol, from the grid size $M * N$, the hop size H, and the set of up sensors in the grid, we can compute the *status* (whether reachable or unreachable) of each up sensor in the protocol. The status of an up sensor u is *reachable* iff sensor u is the base station (0,0), or the protocol has an up sensor v such that v is an element of $P.u$ and the status of v is reachable. Otherwise, the status of sensor u is *unreachable*. Note that a down sensor has no status, and if the base station is up, then its status is always reachable.

A configuration of the protocol is *legitimate* iff the following two conditions hold for every up sensor u in the protocol.

- *If sensor u is reachable, sensor u is in a c-state in the configuration.*
- *If sensor u is unreachable, sensor u is in a d-state and* $trc.u = 0$ *in the configuration.*

Lemma 5.1: *Starting from any legitimate configuration, for every reachable sensor u, the execution of any action in sensor u yields a legitimate configuration.*

Proof. The protocol has two cases to consider. In the first case, the executed action is the receiving action in sensor u. In this case, if sensor u receives a

connected message from one of its potential parents, $trc.u$ is set to $tmax$. Otherwise, sensor u drops the message and does nothing. Therefore, the receiving action in u yields a legitimate configuration. In the second case, the executed action is the timeout action in sensor u. In this case, sensor u first decreases $trc.u$ by one. However, since u is reachable, there exists at least one up sensor v such that v is an element of $P.u$ and the status of v is reachable. Thus, u will receive a connected message from a sensor that is an element of $P.u$ before $trc.u$ becomes 0, and so $trc.u$ remains to be bigger than 0 (by assumptions A3 and A4). Second, u sends a connected message (since $trc.u > 0$). If there is a sensor v where $u \in P.v$ and $trc.v = 1$, then v receives the connected message sent by u and sets $trc.v$ to $tmax$ (by assumption A4). Third, u chooses a random value for $timer.u$. If $trc.u = 1$, u chooses a value for $timer.u$ according to assumption A3, and so u remains in a c-state. Therefore, the timeout action in u yields a legitimate configuration. □

Lemma 5.2: *Starting from any legitimate configuration, for every unreachable sensor u, the execution of any action in sensor u yields a legitimate configuration.*

Proof. The protocol has two cases to consider. In the first case, the executed action is the receiving action in sensor u. In this case, since sensor u is unreachable, the protocol has no up sensor v such that v is an element of $P.u$ and the status of v is reachable. Thus, sensor u always drops the received message, and

does nothing. Therefore, the receiving action in u yields a legitimate configuration. In the second case, the executed action is the timeout action in sensor u. In this case, $trc.u$ remains 0 by executing the statement "max(trc-1,0)". Therefore, the receiving action in u yields a legitimate configuration. □

Theorem 5.1: *(Closure) Starting from any legitimate configuration, the protocol is guaranteed to maintain a legitimate configuration.*

Proof. The proof follows from the proofs of Lemmas 5.1, and 5.2. □

The following definitions are useful to prove the convergence of the protocol. We define the *connection graph* C of the protocol as a directed graph that satisfies the following two conditions. First, each node in graph C represents a distinct sensor in the set of reachable sensors in the protocol. Second, each directed edge (u, v) from a node u to a node v in graph C indicates that sensor v is a potential parent of sensor u. We define the height of a connection graph C as the maximum number of edges in any path from any node to the base station in graph C. Let the height of graph C be K.

We define the *disconnection graph* D of the protocol as a directed graph that satisfies the following two conditions. First, each node in graph D represents a distinct sensor in the set of unreachable sensors in the protocol. Second, each directed edge (u, v) from a node u to a node v in graph D indicates that sensor v is a potential parent of sensor u. We define the height of a disconnec-

tion graph D as the maximum number of nodes in any path from any node to another node in graph D. Let the height of graph D be L.

Next, we show that starting from any illegitimate configuration, our protocol is guaranteed to reach a legitimate configuration within a finite time period.

Lemma 5.3: *Starting from any illegitimate configuration, every reachable sensor u is guaranteed to be in a c-state within $K * tmax * rmax$ time units.*

Proof. Let the base station be node u_0 in the connection graph C of the protocol. If u_0 is up, u_0 sends a connected message periodically. (Note that if u_0 is down, the connection graph of the protocol does not exist.) Thus, for every node u_1 where there exists edge (u_1,u_0) in graph C, u_1 is guaranteed to receive at least one connected message from u_0, and reach a c-state by setting $trc.u$ to $tmax$ within $tmax * rmax$ time units (by assumptions A3 and A4). Moreover, u_1 is guaranteed to start sending a connected message periodically within $tmax * rmax$ time units. Assume that for every node u_{K-1} where there exist edges (u_i,u_{i-1}), $1 \leq i \leq K-1$, in graph C, u_{K-1} is guaranteed to reach a c-state and start sending a connected message periodically within $(K-1) * tmax * rmax$ time units. For every node u_K where there exist edges (u_i,u_{i-1}), $1 \leq i \leq K$, in graph C, u_K is guaranteed to receive at least one connected message from any u_{K-1}, and reach a c-state within $K * tmax * rmax$ time units (by assumptions A3 and A4). Therefore, every node in graph C reaches a c-state within $K * tmax * rmax$ time units, if the height of graph C

is K. □

Lemma 5.4: *Starting from any illegitimate configuration, for every unreachable sensor u, it is guaranteed that sensor u is in a d-state and trc.u = 0 within* $L * tmax * rmax$ *time units.*

Proof. Every unreachable sensor u is always in a d-state. Therefore, we only need to show that $trc.u$ becomes 0 within a finite time period. In graph D, there exists at least one node that has no outgoing edge, since all nodes in graph D are unreachable. For every node u_0 where there exists no outgoing edge from u_0 in graph D, u_0 will not receive any connected message from a node v where there exists edge $(u_0,\ v)$ in graph D. Thus, it is guaranteed that $trc.u_0$ becomes 0 within $tmax * rmax$ time units. Assume that for every node u_{L-2} where there exist edges (u_i,u_{i-1}), $1 \leq i \leq L-2$, in graph D, it is guaranteed that $trc.u_{L-2}$ becomes 0 within $(L-1) * tmax * rmax$ time units. Since u_{L-2} does not send a connected message, for every node u_{L-1} where there exist edges (u_i,u_{i-1}), $1 \leq i \leq L-1$, in graph D, u_{L-1} will not receive any connected message from any u_{L-2}. Thus, it is guaranteed that $trc.u_{L-1}$ becomes 0 within $L * tmax * rmax$ time units. Therefore, for every node u in graph D, it is guaranteed that u is in a d-state and $trc.u = 0$ within $L * tmax * rmax$ time units, if the height of graph D is L. □

Theorem 5.2: *(Convergence) Starting from any illegitimate configuration, the protocol is guaranteed to reach a legitimate configuration within max(K ∗*

$tmax * rmax,\ L * tmax * rmax)$ *time units.*

Proof. The proof follows from the proofs of Lemmas 5.3, and 5.4. □

5.4 The Routing Protocol with Foster Parents

According to the LGR protocol in Section 5.2, a sensor has a parent in the routing tree as long as this sensor keeps on receiving connected messages from one or more of its potential parents. Thus, if at least one of the potential parents of a sensor is up and connected to the routing tree, the sensor remains connected to the tree. Unfortunately, it is possible that all the potential parents of a sensor fail-stop. When this happens, the sensor no longer receives any connected messages from any of its potential parents, and so it becomes disconnected from the routing tree.

To solve this problem, we extend the LGR protocol such that a sensor remains connected to the routing tree even if all its potential parents have fail-stopped as follows. When a sensor (i,j) has no parent and receives a connected message from some sensor (i',j'), which is not a potential parent of sensor (i,j), sensor (i,j) makes sensor (i',j') its *foster parent* in the routing tree. In this case, sensor (i,j) becomes connected to the tree and it can route the data messages to the base station, but it does not send any connected(i,j) messages. (The reason for this last restriction is to prevent any subset of sensors from forming a directed cycle of foster parent relationships.)

When a sensor (i,j) is connected to the routing tree via a foster parent (i',j'), and receives a connected(i'',j'') message from a sensor (i'',j''), then sensor (i,j) makes sensor (i'',j'') its parent or its foster parent (depending on whether (i'',j'') is a potential parent of (i,j)) in the routing tree.

When a sensor (i,j) is connected to the routing tree via a foster parent, but does not receive any connected message for a time period of $tmax * T$ time units, it becomes disconnected from the tree and no longer forwards data messages to the base station.

A specification for sensor (i,j), where $i \neq 0$ or $j \neq 0$, is given in Fig. 5.6. (Note that a specification for the base station is identical to the one in Section 5.2.)

5.5 Stabilization of the Protocol with Foster Parents

In this section, we sketch a proof that the protocol in Section 5.4 is self-stabilizing. This proof is based on assumptions A1 and A2 discussed in Section 5.3, and the following two assumptions.

B3) For every up sensor u, if the value of trc in sensor u is 1, then sensor u chooses a value for $timer.u$ such that the value of $timer.u$ is larger than the value of $timer.v$ for any up neighbor v of sensor u.

B4) For every up sensor u, if the value of trc in sensor u is 0 or 1, and any neighbor of sensor u sends a connected message, then sensor u receives

```
sensor (i,j)    // a sensor (i,j) in an M*N grid where i=!0 or j!=0

const P         : set of potential parents of sensor (i,j),
      rmin,rmax : integer,    // min and max interval whose avg is T
      tmax      : integer

var   pid       : a sensor (i',j') where i!=i' or j!=j', // parent identifier
      trc       : 0..tmax,    // time to remain connected
      r         : rmin..rmax, // random interval to send next connected msg
      x         : 0..M-1,
      y         : 0..N-1

begin
   rcv connected(x,y) ->
       if ((x,y) in P) or (trc=0) or (trc>0 and !(pid in P)) ->
               pid := (x,y);         // choose new parent
               trc := tmax
       [] !((x,y) in P) and (trc>0) and (pid in P) -> skip
       fi

[] time-out expires ->   trc := max(trc-1,0);
                         if trc>0 and (pid in P) -> send connected(i,j);
                         [] trc=0 or !(pid in P) -> skip
                         fi; r := rand; time-out after r
end
```

Figure 5.6: Specification of sensor (i,j) in the LGR protocol with foster parents

the connected message.

Note that the proof in this section is similar to that in Section 5.3. However, the proof in this section needs to consider the cases where some sensors in the protocol have foster parents.

In the protocol, a *state* of a sensor u is defined by the value of variable *pid*, the value of variable *trc*, and the value of variable *timer.u* in sensor u, and the state of each up neighbor v of sensor u. Sensor u is in one of the following three states:

i. Sensor u is in a *c-state* iff u is the base station, or at this state
$pid.u \in P.u$ and (($trc.u > 1$ and there exists an up potential parent v of u where v is in a c-state) or
($trc.u = 1$ and there exists an up potential parent v of u where v is in a c-state and $timer.u > timer.v$))

ii. Sensor u is in a *j-state* iff at this state
$pid.u \notin P.u$ and (($trc.u > 1$ and there exists an up neighbor v of u where v is in a c-state) or
($trc.u = 1$ and there exists an up neighbor v of u where v is in a c-state and $timer.u > timer.v$))

iii. If sensor u is neither in a c-state nor in a j-state, then sensor u is in a *d-state*.

Regardless of the initial configuration of the protocol, from the grid size $M * N$, the hop size H, and the set of up sensors in the grid, we can compute the *status* (whether reachable, weakly reachable, or unreachable) of each up sensor in the protocol. The status of an up sensor u is *reachable* iff sensor u is the base station (0,0), or the protocol has an up sensor v such that v is an element of $P.u$ and the status of v is reachable. The status of sensor u is *weakly reachable* iff sensor u is not reachable, and the protocol has an up sensor v such that v is an element of $Ngh.u$ and the status of v is reachable, where $Ngh.u$ is the set of up sensors that are neighbors of sensor u and are not elements of $P.u$. Otherwise, the status of sensor u is *unreachable*.

A configuration of the protocol is *legitimate* iff the following three conditions hold for every up sensor u in the protocol.

- *If sensor u is reachable, sensor u is in a c-state in the configuration.*
- *If sensor u is weakly reachable, sensor u is in a j-state in the configuration.*
- *If sensor u is unreachable, sensor u is in a d-state and* $trc.u = 0$ *in the configuration.*

Lemma 5.5 : *Starting from any legitimate configuration, for every reachable sensor u, the execution of any action in sensor u yields a legitimate configuration.*

Lemma 5.6 : *Starting from any legitimate configuration, for every weakly reachable sensor u, the execution of any action in sensor u yields a legitimate*

configuration.

Lemma 5.7 : *Starting from any legitimate configuration, for every unreachable sensor u, the execution of any action in sensor u yields a legitimate configuration.*

The proofs of Lemmas 5.5, 5.6, and 5.7 are similar to those of Lemmas 5.1 and 5.2.

Theorem 5.3: *(Closure) Starting from any legitimate configuration, the protocol is guaranteed to maintain a legitimate configuration.*

Proof. The proof follows from the proofs of Lemmas 5.5, 5.6, and 5.7. □

We define the connection graph C of the protocol and the height of the connection graph, K, as they are defined in Section 5.3. We also define the disconnection graph D of the protocol and the height of the disconnection graph, L, as they are defined in Section 5.3. Next, we show that starting from any illegitimate configuration, our protocol is guaranteed to reach a legitimate configuration within a finite time period.

Lemma 5.8 : *Starting from any illegitimate configuration, every reachable sensor u is guaranteed to be in a c-state within $K * tmax * rmax$ time units.*

Proof. This proof is similar to that of Lemma 5.3. □

Lemma 5.9 : *Starting from any illegitimate configuration, every weakly reachable sensor u is guaranteed to be in a j-state within $(K+1) * tmax * rmax$*

time units.

Proof. By Lemma 5.8, every reachable sensor is guaranteed to reach a c-state, and start sending a connected message periodically within $K * tmax * rmax$ time units. Since sensor u is weakly reachable, there exists at least one up sensor v such that v is an element of $Ngh.u$, and the status of v is reachable. Thus, every sensor u is guaranteed to receive at least one connected message from a sensor that is an element of $Ngh.u$, and reach a j-state within $(K + 1) * tmax * rmax$ time units (by assumptions B3 and B4). □

Lemma 5.10 : *Starting from any illegitimate configuration, for every unreachable sensor u, it is guaranteed that sensor u is in a d-state and* $trc.u = 0$ *within* $(L + 1) * tmax * rmax$ *time units.*

Proof. Every unreachable sensor u is always in a d-state. Therefore, we only need to show that $trc.u$ becomes 0 within a finite time period. For every node u_0 where there exists no outgoing edge from u_0 in graph D, u_0 will not receive any connected message from a node v where there exists edge (u_0, v) in graph D. Thus, it is guaranteed that if $trc.u_0 > 0$, $pid.u_0$ is not an element of $P.u_0$ within $tmax * rmax$ time units, and also $pid.u_0$ cannot be an element of $P.u_0$ from this point. Similar to the proof of Lemma 5.4, we can prove that for every node u in graph D, it is guaranteed that if $trc.u > 0$, $pid.u$ is not an element of $P.u$ within $L * tmax * rmax$ time units, and also $pid.u$ cannot be an element of $P.u$ from this point. Thus, every node u in graph D will not

send a connected message, and so will not receive any connected message from any node in graph D. Also node u cannot receive any connected message from a node that is not in graph D. Thus, for every node u in graph D, it is guaranteed that u is in a d-state and $trc.u = 0$ within $(L+1) * tmax * rmax$ time units, if the height of graph D is L. □

Theorem 5.4 : *(Convergence) Starting from any illegitimate configuration, the protocol is guaranteed to reach a legitimate configuration within max($(K+1) * tmax * rmax$, $(L+1) * tmax * rmax$) time units.*

Proof. The proof follows from the proofs of Lemmas 5.8, 5.9, and 5.10. □

5.6 Simulation and Experimental Results

In this section, we first show the effectiveness of the foster parent extension by simulation, and then discuss the experimental results of the LGR protocol.

Simulation results We evaluated the effectiveness of the foster parent extension by simulation. We considered a sensor network that is configured into a 10*10 logical grid with a hop size 2. We assumed that a percentage of the sensors in this network have fail-stopped, and we computed how many of the remaining sensors are still connected to the routing tree via parents or via

foster parents, and how many sensors become disconnected from the routing tree. The results of these simulations are shown in Figures 5.7 and 5.8, and in Table 5.1, where each result represents the average value over 100 simulations.

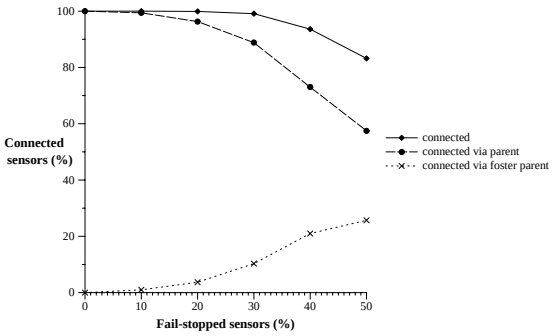


Figure 5.7: Percentage of sensors that are connected to the tree vs. percentage of fail-stopped sensors

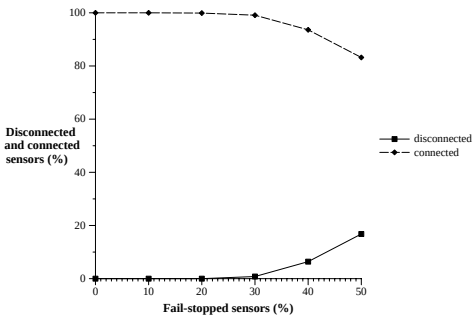


Figure 5.8: Percentage of sensors that are disconnected from the tree vs. percentage of fail-stopped sensors

Table 5.1: Number of fail-stopped sensors vs. Number of connected sensors in 10*10 grid when H=2

# Fail-stopped sensors	# Connected sensors	# Disconnected sensors
0	100	0
10	89	1
20	79	1
30	68	2
40	56	4
50	42	8

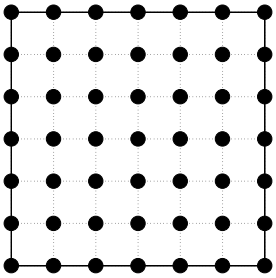
The good news from this study is that even if 50% of the sensors in the network have fail-stopped, about 84% of the remaining sensors remain

connected to the tree. (Of those, 57% are connected via parents and 27% are connected via foster parents.) Only 16% of the remaining sensors become disconnected from the tree. These figures demonstrate that the foster parent extension is highly effective especially when a large fraction of the sensors in the network fail-stop.

Experimental results The LGR protocol has been used for the field experiment "A Line in the Sand" [4], where around 100 MICA2 motes were deployed to monitor a field so that intruders (e.g., tanks, cars, and civilians) can be detected, classified, and tracked. This experiment showed that the LGR protocol is able to reliably route data messages from every sensor across the network to the base station and thus provides the foundation for precise target detection, classification, and tracking. The LGR protocol also has been applied to the large scale field experiment "ExScal" [7], where about 1000 XSMs (for eXtreme Scale Motes) and 200 XSSs (for eXtreme Scale Stargates) were deployed to detect and track intruders. In this experiment, again the LGR protocol has been able to successfully provide reliable message delivery with the delay less than 2 seconds.

To study in more depth the property of the LGR protocol in forwarding packets from different locations, we set up a testbed where 49 MICA2 motes [1] are deployed in a grass field (see Fig. 5.9(a)), forming a 7*7 grid (see Fig. 5.9(b)) with a 5-feet separation between neighboring grid points. The base station (0,0) is the mote at the left-bottom corner of the grid.

(a) Testbed environment



(b) Grid topology

Figure 5.9: The network topology of the testbed

For our experiments, we chose the hop size H to be 2 and T to be 20 seconds. To have a valid logical grid, the power level of each sensor was assigned 9 (out of the range 1..255). Based on this setup, we assigned each sensor in the testbed an identifier so that each sensor can compute the identifiers of its potential parents by using the algorithm in Section 5.1. In this testbed, the average number of hops from a sensor to the base station is around 3.3.

In each experiment, we used the traffic trace from the field experiment "A Line in the Sand" [4] to simulate the network load when events occur. The traffic trace corresponds to an event where each mote except the base station generates two data messages whose interval is between 3 and 4.5 seconds, and overall 96 data messages are generated. The cumulative distribution of the number of data messages that are generated by the sensors in the network during the event is shown as Fig. 5.10. Each performance result discussed in this section represents the average value over 10 runs of this trace.

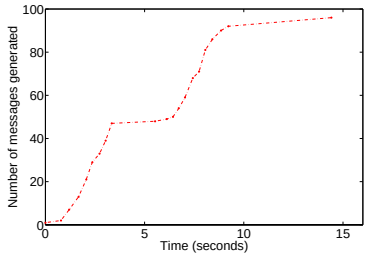


Figure 5.10: The traffic distribution

We evaluate the performance of a routing protocol by the following four metrics:

- *Total delivery ratio*: the ratio of the total number of unique data messages received by the base station to the total number of (unique) data messages generated by all sensors in the network.

- *Individual delivery ratio*: the ratio of the number of unique data messages received by the base station from a particular sensor to the number of data messages generated by that particular sensor.

- *Delay*: the average time taken for a data message to be received by the base station after the data message is generated.

- *Goodput*: the number of unique data messages received by the base station divided by the interval between the time the first data message is generated and the time the last data message is received by the base station. Note that the goodput reflects how fast data messages are pushed from the network to the base station.

First, we ran experiments of the LGR protocol with the default TinyOS queue management component "QueuedSend" which simply retransmits a message up to a certain number of times until the acknowledgment of the message is received [2]. Fig. 5.11 shows the individual delivery ratio when the maximum number of per hop retransmissions is 0 (that is, no retransmission). The individual delivery ratio of each sensor reduces gradually as the number of hops from a sensor to the base station increases, and the total delivery ratio is 72%. The LGR protocol provides reliable uniform delivery of data messages from different locations. This is because the LGR protocol avoids unreliable links to build a reliable routing tree. Also it balances the load over the network and avoids causing severe contention or congestion at certain network locations. Table 5.2 also shows the results of the LGR protocol with QueuedSend when the maximum number of per hop retransmissions is up to 2.

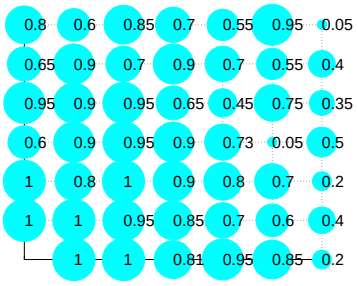


Figure 5.11: Individual delivery ratios of LGR with QueuedSend

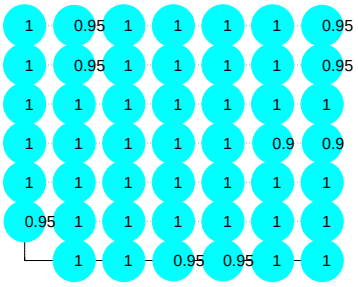


Figure 5.12: Individual delivery ratios of LGR with RBC

To further improve the delivery ratio, we used the LGR protocol with

Table 5.2: Performance of LGR with QueuedSend

	Total delivery ratio	Delay	Goodput
0 retransmission	72%	0.09 seconds	4.52 messages/second
1 retransmission	77.6%	0.11 seconds	4.83 messages/second
2 retransmission	81%	0.12 seconds	4.82 messages/second

a transport protocol RBC developed in [73]. RBC replicates the acknowledgement for a received message using a window-less block acknowledgement scheme, and schedules message retransmissions to alleviate contention caused by them. Thus, lost messages are detected reliably and retransmitted at appropriate time without introducing much additional contention or congestion to the network. We ran experiments of the LGR protocol with RBC where the maximum number of per hop retransmissions is 2, and the results are shown in Fig. 5.12 and Table 5.3. From Fig. 5.12, we observe that the individual delivery ratio of each sensor is almost 100%, and the total delivery ratio is 98.8%. The delay increases since lost data messages are deferred in retransmission. However, this delay does not decrease the goodput which is more important to sensor network applications. Moreover, the delay is good enough for some typical sensor network applications, such as intrusion detection and tracking [4, 7]. The goodput reaches 6.45 messages/second. Note that the optimal goodput for the trace is 6.66 messages/second, when the delay of each data message is 0 and all the data messages in the trace are received by the base station.

Table 5.3: Performance of LGR with RBC

Total delivery ratio	Delay	Goodput
98.8%	1.2 seconds	6.45 messages/second

5.7 Advantages and Limitations

The LGR protocol has the following advantages (over other routing protocols in sensor networks): simplicity, load balancing, fast fault recovery, security without cryptography, and supporting soft real-time applications. Next, we discuss each of the advantages of the protocol.

i) Simplicity The LGR protocol is simple and so it consumes a small percentage of the network resources (specially compared to traditional distance-vector routing protocols). In the LGR protocol, each sensor is required to store no more than 10–15 bytes of routing information that includes P, pid, and trc, and to send a connected message that has only one message field every 20 seconds (or less frequently).

ii) Load balancing Each sensor distributes data traffic over all its potential parents, and so the data items from the same source sensor are likely to follow different paths to the base station. This feature can also help for sensors to avoid severe congestion and reduce message collision, when a burst of sensing events occurs in the network.

iii) Fast fault recovery When the current parent of a sensor fail-stops, the sensor will replace this parent as soon as it receives a connected message from another potential parent. Thus, sensors in the network can recover quickly from the situation where their parents fail-stop.

iv) Security without cryptography In the distance vector routing protocol, an adversary sensor i can tempt many sensors to choose i to be their parents in the routing tree by advertising a very small distance, say 1, to the base station. The adversary then can drop all the data messages that it receives from these sensors. However, in the LGR protocol, each sensor has a set of potential parents in the routing tree and keeps changing its parent whenever it receives a connected message from another potential parent. Thus, the adversary cannot make the sensor choose the adversary to be the parent of the sensor and then drop all data messages from that sensor, unless all the (legitimate) potential parents of the sensor fail-stop. The LGR protocol provides this security feature without encrypting and decrypting connected messages.

v) Supporting soft real-time applications The LGR protocol inherently bounds the number of hops that need to be traversed by a data message from any sensor to the base station. Thus, the delivery delay from any sensor to the base station can be predicted accurately in many soft real-time applications, as discussed in [13]. The number of hops C from a sensor (i,j) that has a parent to the base station is bounded as follows.

$$C \leq max(\lceil \frac{i}{H} \rceil + j, i + \lceil \frac{j}{H} \rceil)$$

On the other hand, the LGR protocol has the following limitations: initial setup requirement, limited communication patterns, limited connectivity, and no support for mobility. Next, we discuss each limitation of the protocol.

i) Initial setup requirement The LGR protocol needs to be set up. Each sensor may need to use GPS or a localization service [36, 61] to compute a distinct identifier in a logical grid, specially for a large network. Also we perform off-line experiments to estimate link quality so that we use these results to identify reliable links. However, once this setup is over, the sensors can take all the advantages that the LGR protocol provides.

ii) Limited communication patterns The LGR protocol does not support every communication pattern that can happen in a network. For example, two sensors in the network cannot exchange data messages unless one of them is the base station. Nevertheless, the protocol provides the two communication patterns that are most commonly used by sensor network applications.

iii) Limited connectivity The LGR protocol limits the connectivity of the sensors in a network such that sensors connected to the tree via foster parents do not send connected messages. Due to this limitation, some sensor in the network may not find a path to the base station, even when there is a possible

path to the base station. However, the probability that this case happens in the network is very small. We showed in Section 5.4 that even when 50% of sensors in the network fail-stop, 84% of the sensors still can be connected to the tree.

iv) No support for mobility The sensors in the logical grid are assigned distinct identifiers such that their identifiers reflect their physical adjacencies. This cannot be maintained in mobile environments. However, in most sensor network applications, the sensors, specially the sensors that participate in routing, are stationary in the network.

5.8 Related Work

Several experimental studies have been done to understand the nature of dynamic and lossy wireless sensor networks. Ganesan et al. [25] performed an experimental study on simple flooding, and reported the complex behavior of flooding, such as non-uniform flood propagation, long links, and asymmetric links. Zhao and Govindan [76] investigated packet delivery performance in different indoor and outdoor environments. They showed that a gray area where reception rate varies dynamically exists over relatively long links, and that the probability of packet loss changes over different traffic patterns. Similar study was performed in [14]. They observed that there is no clear correlation between packet delivery and distance over relatively long links. The link quality

estimated by periodic beacon exchanges usually reflects the link quality in the absence of bursty data traffic in sensor networks. Zhang et al. [74] showed that a link quality changes significantly when the traffic pattern within the network changes. Thus, it is hard to accurately estimate link quality, especially under bursty traffic, since different bursts of traffic usually occur sparsely, and each burst may finish within a short period.

Woo et al. [68] investigated link quality estimation and neighborhood management in dynamic and lossy sensor networks. They developed a routing protocol, where each sensor selects a path with the minimal number of retransmissions based on the estimated link quality. Seada et al. [60] proposed link selection strategies based on estimated link quality for geographic routing protocols. Under bursty traffic, the estimated link quality may not be accurate and stable, resulting in low message delivery rate.

In geographic routing protocols surveyed in [63], physical location information is used to route data messages. Each node maintains its one-hop neighborhood information, and forwards a message to one of its neighbors that is closer to the destination. As discussed in [60], these protocols can select unreliable long links to forward a message to the destination. He et al. [31] proposed a real-time routing protocol that utilizes location information. To support end-to-end soft real-time communication, each sensor dynamically estimates single hop delay with all its neighbors, and uses the estimated delay to select the next hop that satisfies a desired speed. Thus, the delivery delay

is proportional to the distance between the source and the destination. The major focus of the LGR protocol is to provide reliable delivery (of bursty data traffic) to the base station, yet our protocol supports soft real-time applications using the logical grid structure. The experimental results showed that the LGR protocol has successfully delivered data messages with acceptable delay for a tracking application in [4, 7].

Routing protocols were proposed to build virtual coordinates without location information and route messages based on virtual coordinates [13, 53, 56]. The computation of virtual coordinates is usually based on hop counts from anchors. In [22], sensors in a network form a layered multi-hop structure based on hop counts to the base station, and each sensor forwards messages to one of its neighbors that have a smaller hop count than itself. These protocols build a routing tree based on hop counts to certain sensors. Therefore, they take advantage of all long links in sensor networks and they may use unreliable long links.

A self-stabilizing algorithm was proposed to maintain a spanning tree in a network [6]. This algorithm requires a pre-specified bound on the network size, and it is not guaranteed to stabilize within a time period proportional to the diameter of the network. On the other hand, the LGR protocol stabilizes, without any prior knowledge of the network size or diameter, within a time period proportional to the diameter of the network.

The algorithms presented in [8, 11, 26, 44] were designed to localize the

impact of faults in routing. The LGR protocol does not consider fault containment, but the mechanisms used by these papers can be applied to the LGR protocol. Detailed study of this is beyond the scope of our work.

Chapter 6

Flood Sequencing Protocols in Sensor Networks

Flood is a communication primitive that can be used by the base station of a sensor network to send a copy of a message to every sensor in the network. The execution of a flood starts by the base station sending a message to all its neighbors. When a sensor receives a message, the sensor needs to check whether it has received this message for the first time or not. Only if the sensor has received the message for the first time, the sensor keeps a copy of the message and may forward the message to all its neighbors. Otherwise, the sensor discards the message.

To distinguish between "fresh" flood messages that a sensor should keep and "redundant" flood messages that a sensor should discard, the base station selects a sequence number and attaches it to a flood message before the base station broadcasts the message. When a sensor receives a flood message, the sensor determines based on the sequence number in the received message if the message is fresh or redundant. The sensor accepts the message if it is fresh and discards the message if it is redundant. We call a protocol that uses

sequence numbers to distinguish between fresh flood messages and redundant flood messages a *flood sequencing protocol*.

In a flood sequencing protocol, when a fault corrupts the sequence numbers stored in some sensors in a sensor network, the network can become in an illegitimate state where the sensors discard fresh flood messages and accept redundant flood messages. Therefore, a flood sequencing protocol should be designed such that if the protocol ever reaches an illegitimate state due to some fault, the protocol is guaranteed to converge back to its legitimate states where every sensor accepts every fresh flood message and discards every redundant flood message.

In this chapter, we discuss a family of four flood sequencing protocols. They are a *sequencing free* protocol, a *linear sequencing* protocol, a *circular sequencing* protocol, and a *differentiated sequencing* protocol. We analyze the stabilization properties of these four protocols. For each of the protocols, we first compute an upper bound on the convergence time of the protocol from an illegitimate state to legitimate states. Second, we compute an upper bound on the number of fresh flood messages that can be discarded by each sensor during the convergence. Third, we compute an upper bound on the number of redundant flood messages that can be accepted by each sensor during the convergence.

The rest of this chapter is organized as follows. In Section 6.1, we discuss related work and motivation of the flood sequencing protocols. In Section 6.2, we give an overview of a flood protocol. We present the four flood

sequencing protocols and analyze their stabilization properties in Sections 6.3, 6.4, 6.5, and 6.6. In Section 6.7, we show the simulation results of these protocols.

6.1 Motivation

The practice of using sequence numbers to distinguish between fresh and redundant flood messages has been adopted by most flood protocols in the literature. In other words, most flood protocols "employ" some flood sequencing protocols to distinguish between fresh and redundant flood messages. A flood sequencing protocol can be designed in various ways, depending on several design decisions such as how the next sequence number is selected by the base station, how each sensor determines based on the sequence number in a received message if the received message is fresh or redundant, and what information the base station and each sensor stores in its local memory. Unfortunately, flood sequencing protocols have so far not been studied well in the literature. They have been used without full investigation of their design decisions.

The flood protocols discussed in [25, 45, 57, 64] assume that when a sensor receives a flood message, the sensor can figure out whether the sensor has received this message for the first time or not, without specifying any mechanism to achieve this. In [33, 55], it was suggested to associate a sequence number with each flood message, but any details on how sequence numbers are

used by sensors (i.e. the design decisions of their flood sequencing protocols) were not specified. The flood protocols discussed in [39, 67] propose to attach a unique identifier to each flood message and make each sensor maintain a list of identifiers that the sensor has received recently. Similarly, it was suggested in [65] that each sensor maintains a list of flood messages received by the sensor recently. However, any details such as how many identifiers or messages each sensor maintains and when a sensor deletes an identifier or a message from the list were not discussed.

A flood sequencing protocol is important, since the fault tolerance property of a sensor network is affected by a flood sequencing protocol used in the network. When a fault corrupts the sequence number stored in some sensor in the network, the sensor may discard fresh flood messages and accept redundant flood messages. The number of fresh flood messages discarded by the sensor and the number of redundant flood messages accepted by the sensor, before the network reaches a legitimate state, are different depending on which flood sequencing protocol is used in the network or how a flood sequencing protocol used in the network is designed. Therefore, we need to study various flood sequencing protocols and analyze the stabilization properties of these protocols. The stabilization properties of the flood sequencing protocols are useful for sensor network designers or developers to select a proper flood sequencing protocol that satisfies the needs of a target sensor network.

In practice, a flood sequencing protocol is used with a flood protocol that may use other techniques to improve the performance of flood such as

reliability or efficiency. In this chapter, each of the flood sequencing protocols is described focusing on how sequence numbers are used by sensors, and it is not described as a specific flood protocol. Note that the stabilization property of a flood protocol is affected by that of a flood sequencing protocol used in the flood protocol. If the flood protocol does not maintain any extra state such that it is based on probability [33, 57], the stabilization property of the flood protocol is the same as that of the used flood sequencing protocol. If the flood protocol maintains extra state such that it is based on neighbor information [46, 64], the stabilization property of the flood protocol also depends on how the extra state in each sensor is stabilized.

6.2 Overview of a Flood Protocol

In this section, we give an overview of a flood protocol that is used with our flood sequencing protocols. Consider a network that has n sensors. In this network, sensor 0 is the base station and can initiate floods over the network. To initiate the flood of a message, sensor 0 sends a message of the form data($hmax$), where $hmax$ is the maximum number of hops to be made by this data message in the network.

If sensor 0 initiates one flood and shortly after initiates another flood, some forwarded messages from these two floods can collide with one another causing many sensors in the network not to receive the message of either flood, or (even worse) not to receive the messages of both floods.

To prevent message collision across consecutive flood messages, once sensor 0 broadcasts a message, it needs to wait enough time until this message is no longer forwarded in the network, before broadcasting the next message. The time period that sensor 0 needs to wait after broadcasting a message and before broadcasting the next message is called the *flood period*. The flood period consists of f time units. (A lower bound on the value of f is computed below.) Thus, after sensor 0 broadcasts a message, it sets its timeout to expire after f time units in order to broadcast the next message.

When a sensor receives a data(h) message, the sensor decides whether the sensor accepts the message and forwards it as a data($h-1$) message, provided $h > 1$. To reduce the probability of message collision, any sensor u, that decides to forward a message, chooses a random period whose length is chosen uniformly from the range $1..tmax$, and sets its timeout to expire after the chosen random period, so that u can forward the received message at the end of the random period. This random time period is called the *forwarding period*.

Next, we compute the lower bound of the flood period f.

Theorem 6.0:

$$f \geq (hmax - 1) * tmax + 1$$

Proof. When sensor 0 broadcasts a data($hmax$) message at time t, an out-neighbor u of sensor 0 can receive the message at t and choose the maximum

possible value $tmax$ for the forwarding period. At time $t + tmax$, u forwards the message as data($hmax - 1$). Similarly, an out-neighbor u' of sensor u can receive the message at $t + tmax$ and choose $tmax$ for the forwarding period. This forwarding process continues until this message makes $hmax$ hops. Therefore, some sensor u can receive the last data(1) message at time $t + (hmax - 1) * tmax$ in the worst case. Thus, the flood period needs to be at least $(hmax - 1) * tmax + 1$ time units to guarantee that no forwarded messages from two consecutive floods collide with one anther. □

To analyze each of the four flood sequencing protocols, we use the following value for the flood period f.

$$f = hmax * tmax + 1$$

(We choose this value for f, instead of the minimum value $(hmax-1)*tmax+1$, to keep our proofs of the stabilization properties simple.)

Note that the above flood period is computed to guarantee that no two consecutive flood messages ever collide with each other. In a typical execution of the protocol, each sensor chooses its forwarding period at random in the range $1..tmax$, and so most sensors likely receive the flood messages within $(hmax - 1) * tmax/2$ time units, instead of $(hmax - 1) * tmax$ time units. Therefore, the half (or even less) of the flood period may be used without significantly degrading the stabilization property and performance of a flood sequencing protocol.

6.3 First Protocol: Sequencing Free

In this section, we discuss a first flood sequencing protocol where no sequence number is attached to each flood message, and so a sensor cannot distinguish between fresh and redundant flood messages, resulting that the sensor accepts every received message. This protocol is called the *sequencing free* protocol.

To initiate the flood of a new message, sensor 0 sends a data($hmax$) message, and then sets its timeout to expire after f time units to broadcast the next message. A formal specification of sensor 0 is given in Fig. 6.1. Note that sensor 0 does not receive any messages.

```
1:  sensor 0                              {base station}
2:  const  hmax   : integer,              {max hop count}
3:         f      : integer               {flood period}
4:  begin
5:      timeout-expires →  {generate new msg}
6:                         send data(hmax);
7:                         timeout-after f
8:  end
```

Figure 6.1: A specification of sensor 0 in the sequencing free protocol

Each sensor u that is not sensor 0 maintains a variable called new. The value of new is true only when u is in the forwarding period (i.e. u has a flood message that has been received earlier but has not been forwarded yet). When sensor u receives a data(h) message, u always accepts the message. Sensor u

forwards the message as data($h-1$), if $h > 1$ in the received message and $new = false$ in u. A formal specification of sensor u is given in Fig. 6.2. (Each sensor u also maintains a received data message that u will forward later, even though this is not explicitly specified in the specification.)

```
1:  sensor u:1 .. n − 1
2:  const hmax   : integer,          {max hop count}
3:        tmax   : integer           {max forwarding period}
4:  var   h,hlast : 1 .. hmax,       {rcvd,last hop count}
5:        new    : boolean           {true if u has msg to forward}
6:  begin
7:     timeout-expires → if new →        new := false;
8:                                       send data(hlast)
9:                       [] ¬ new →     skip
10:                      fi; timeout-after random(1,tmax)

11:    [] rcv data(h) →   {accept msg}
12:                       if h>1 ∧ ¬ new →   new := true;
13:                                          hlast := h − 1
14:                       [] h≤ 1 ∨ new →    skip
15:                       fi
16: end
```

Figure 6.2: A specification of sensor u in the sequencing free protocol

Note that in all the flood sequencing protocols presented in this chapter, the value of timer.0 is at most f time units, and the value of timer.u is at most $tmax$. This is maintained by the executions of all the protocols.

A state S of the sequencing free protocol is *legitimate* iff either S is a state where the predicate

$$\text{(timer.0= 1)} \wedge \text{(for all } u,\ u \neq 0,\ \text{new.}u = false)$$

holds or S is a state that is reachable from a state, where this predicate holds, by some execution of the protocol.

It follows from this definition that if the protocol is executed starting from a legitimate state, then every time sensor 0 initiates a new flood, previous flood messages (whether initiated by sensor 0 legitimately or other sensors illegitimately due to some fault) are no longer forwarded in the network.

The stabilization property of the sequencing free protocol can be stated by the following three theorems. Theorem 6.1A gives an upper bound on the convergence time of the protocol from an illegitimate state to legitimate states. Theorem 6.1B gives an upper bound on the number of fresh messages that can be discarded by each sensor during the convergence. Theorem 6.1C gives an upper bound on the number of redundant messages that can be accepted by each sensor during the convergence. (In general, the stabilization property of each of the other three protocols can be stated by three theorems: Theorem 6.iA, Theorem 6.iB, and Theorem 6.iC, where i=2,3, and 4.) In proofs below, we use the notation <var>.u to denote the value of variable <var> in a sensor u.

Theorem 6.1A: *In the sequencing free protocol, starting from any illegitimate state, the protocol reaches a legitimate state within* $2 * f$ *time units, and continues to execute within legitimate states.*

Proof. (Sketch) Starting from any state, any flood initiated by some sensor u

due to wrong initial values of *new* and *hlast* in u will be terminated within f time units. This is because sensor u will timeout within $tmax$ time units, and the maximum lifetime of a flood message is $(hmax-1)*tmax$ time units. After all wrongly initiated floods are terminiated, $new.u$ for every sensor u always becomes false when timer.u=1. The value of $timer.0$ becomes 1 again within $2*f$ time units. Thus, the protocol reaches a legitimate state within $2*f$ time units, and continuously stays in legitimate states. □

Theorem 6.1B: *In the sequencing free protocol, starting from any illegitimate state, every sensor discards no fresh message (before the protocol converges to a legitimate state).*

Note that starting from any legitimate state, every sensor discards no fresh message, since the sensor accepts every received message.

Theorem 6.1C: *In the sequencing free protocol, starting from any illegitimate state, every sensor accepts at most* $2*f$ *redundant messages (before the protocol converges to a legitimate state).*

Proof. A sensor u can receive at most one message at each time instant. Thus, in the worst case, u can accept a redundant message at each time instant during the convergence time, and so the maximum number of redundant messages accepted by u until convergence is $2*f$. □

Note that even starting from any legitiamte state, the sensor cannot

distinguish between fresh and redundant flood messages. The number of redundant copies of the same message accepted by a sensor u depends on the value of $hmax$ and the network topology. In worst case, u can accept a redundant copy of the same message at each time instant during the flood period of the message. Thus, starting from any legitimate state, every sensor accepts at most f redundant copies of the same message.

6.4 Second Protocol: Linear Sequencing

In this section, we discuss a second flood sequencing protocol where each flood message carries a unique sequence number that is linearly increased, and so a sensor accepts a flood message that has a sequence number larger than the last sequence number accepted by the sensor. This protocol is called the *linear sequencing* protocol.

```
1:  sensor 0                                  {base station}
2:  const hmax   : integer,                   {max hop count}
3:        f      : integer                    {flood period}
4:  var   slast  : integer                    {last seq number}
5:  begin
6:      timeout-expires →                     {generate new msg}
7:                                            slast := slast + 1;
8:                                            send data(hmax,slast);
9:                                            timeout-after f
10: end
```

Figure 6.3: A specification of sensor 0 in the linear sequencing protocol

Each flood message in this protocol is of the form data(h,s), where field h is the remaining number of hops to be made by this message, and field s is the unique sequence number of this message.

Whenever sensor 0 broadcasts a new message, sensor 0 increases the sequence number of the last message by one, and attaches the increased sequence number to the message. A formal specification of sensor 0 is given in Fig. 6.3.

Each sensor u that is not sensor 0 keeps track of the last sequence number accepted by u in a variable called *slast*. When sensor u receives a data(h, s) message, sensor u accepts the message if $s > slast$, and forwards the message if $h > 1$. A formal specification of sensor u is given in Fig. 6.4.

A state S of the linear sequencing protocol is *legitimate* iff either S is a state where the predicate

$$(\text{timer.0}= 1) \wedge (\text{for all } u,\ u \neq 0,\ \text{new.}u{=}false \wedge \text{slast.}u \leq \text{slast.0})$$

holds or S is a state that is reachable from a state, where this predicate holds, by some execution of the protocol.

It follows from this definition that if the protocol is executed starting from a legitimate state, then every time sensor 0 initiates a new flood, previous flood messages are no longer forwarded in the network, and the new flood message has a sequence number that is larger than every slast.u in the network, so that every u accepts the message.

```
1:  sensor u:1 .. n − 1
2:  const hmax   : integer,          {max hop count}
3:        tmax   : integer           {max forwarding period}
4:  var   h,hlast : 1 .. hmax,       {rcvd,last hop count}
5:        s, slast : integer,        {rcvd,last seq number}
6:        new    : boolean           {true if u has msg to forward}
7:  begin
8:     timeout-expires → if new →          new := false;
9:                                         send data(hlast, slast)
10:                       [] ¬ new →       skip
11:                       fi; timeout-after random(1,tmax)

12:    [] rcv data(h, s) → if s > slast →  {accept msg} slast := s;
13:                          if h>1 →      new := true;
14:                                        hlast := h − 1
15:                          [] h≤ 1 →     skip
16:                          fi
17:                       [] s ≤ slast →   {discard msg} skip
18:                       fi
19: end
```

Figure 6.4: A specification of sensor u in the linear sequencing protocol

Let k be the maximum value between 1 and k', where k' is the maximum difference $slast.u - slast.0$ for any sensor u in the network at an initial state. Note that the value of k is finite but it is unbounded.

Theorem 6.2A: *In the linear sequencing protocol, starting from any illegitimate state, the protocol reaches a legitimate state within* $(k+1) * f$ *time units, and continues to execute within legitimate states.*

Proof. (Sketch) There are two cases to consider. In the first case, initially $slast.0$ is larger than or equal to each $slast.u$ in the network, so that $k' < 1$ and $k = 1$. In the second case, initially $slast.0$ is less than some $slast.u$ in the network, so that $k' \geq 1$ and $k \geq 1$. In the first case of $k' < 1$ and $k = 1$, the proof is similar to that of Theorem 6.1A. The protocol reaches a legitimate state within $2 * f$ time units. Consider the second case of $k' \geq 1$ and $k \geq 1$. Let s be the value of $slast.0$ at the initial state. After f time units, $new.u$ for every sensor u always becomes false when timer.0=1. When the value of timer.0 becomes 1 again, say in time unit $(t-1,t)$, within $2 * f$ time units, $slast.0$ is at least $s+1$. Sensor 0 broadcasts a message with sequence number $s+2$ at t. If $k' = 1$, then $s+2$ is larger than each $slast.u$ in the network. Thus, in this case, the protocol reaches a legitimate state within $2 * f$ time units. If $k' > 1$, then sensor 0 broadcasts a message with sequence number $s+3$ at $t+f$ and so on. Finally at $t+(k-2) * f$, sensor 0 broadcasts a message with sequence number $s+k$ and the flood of this message will be terminated by $t+(k-1) * f$. Thus, the protocol reaches a legitimate state within $(k+1) * f$

time units, and continuously stays in legitimate states. □

Theorem 6.2B: *In the linear sequencing protocol, starting from any illegitimate state, every sensor discards at most* $(k+1) * f$ *fresh messages (before the protocol converges to a legitimate state).*

Theorem 6.2C: *In the linear sequencing protocol, starting from any illegitimate state, every sensor accepts at most* $n-1$ *redundant messages (before the protocol converges to a legitimate state).*

Proof. (Sketch) Assume that the protocol starts from a state where $new.u$ for every sensor u is true. In this case, every sensor u can initiate a flood of the previous accepted message. Thus, sensor u can accept at most $n-1$ redundant messages from every other sensor in the network. □

The linear sequencing protocol requires sensors to use unbounded sequence numbers. Thus, this protocol is very expensive to implement for sensor networks that have limited resources. However, once the protocol starts its execution from any legitimate state, every sensor accepts every fresh message and discards every redundant message under any degree of message loss.

6.5 Third Protocol: Circular Sequencing

In this section, we discuss a third flood sequencing protocol where each flood message carries a sequence number that is circularly increased within a limited range, and so a sensor accepts a flood message that has a sequence number "logically" larger than the last sequence number accepted by the sensor. This protocol is called the *circular sequencing* protocol.

Each flood message is augmented with a sequence number that has a value in the range 0 .. $smax$, where $smax > 1$. We assume that $smax$ is an even number (to keep our presentation simple).

Whenever sensor 0 broadcasts a new message, sensor 0 increases the sequence number of the last message by one circularly within the range 0 .. $smax$, and attaches the increased sequence number to the message. The timeout action of sensor 0 in this protocol is modified as Fig. 6.5.

```
1:    timeout-expires →  {generate new msg}
2:                       slast := (slast + 1) mod (smax+1);
3:                       send data(hmax,slast);
4:                       timeout-after f
```

Figure 6.5: Timeout action of sensor u in the circular sequencing protocol

From the viewpoint of each sequence number s in the range 0 .. $smax$, the range can be divided into two subranges, where one subrange consists of the sequence numbers that are logically "smaller" than s, and the other

subrange consists of the sequence numbers that are logically "larger" than s. Thus, sequence number s has $\frac{smax}{2}$ numbers logically smaller than it and $\frac{smax}{2}$ numbers logically larger than it. For example, if $smax = 8$, number 0 is logically smaller than 1, 2, 3, and 4, and is logically larger than 5, 6, 7, and 8.

When a sensor u receives a data(h, s) message, sensor u checks if s is logically larger than $slast$. Sensor u calls the function "Larger$(s, slast)$" that returns true if s is logically larger than $slast$, and otherwise returns false. Sensor u accepts the message if Larger$(s, slast)$ returns true, and forwards it if $h > 1$. The timeout action of sensor u is identical to the one in the linear sequencing protocol in Fig. 6.4, and the receiving action of sensor u is modified as Fig. 6.6.

```
1:    [] rcv data(h, s) →
2:        if Larger(s,slast) →     {accept msg} slast := s;
3:                                 if h>1 →     new := true;
4:                                                 hlast := h − 1
5:                                 [] h≤ 1 →    skip
6:                                 fi
7:        [] ¬Larger(s,slast) →    {discard msg} skip
8:        fi
```

Figure 6.6: Receiving action of sensor u in the circualr sequencing protocol

To prove the stabilization property of the circular sequencing protocol, we make an assumption of bounded message loss as follows:

- *Bounded message loss*: Starting from any state, if sensor 0 broadcasts

$\frac{smax}{2}$ consecutive flood messages, then every sensor in the network receives at least one of those flood messages.

Two explanations concerning the above assumption are in order. First, the protocol may not be self-stabilizing without any bound on message loss. For example, consider a scenario where $smax$=8. Assume that sensor 0 sends a flood message with sequence number 0 and a sensor u accepts the message. If sensor u does not receive the next 4 (i.e. $\frac{smax}{2}$) consecutive messages with sequence numbers 1, 2, 3 and 4, and later receives a fresh message with sequence number 5, it discards the message since sequence number 5 is not logically larger than sequence number 0. Sensor u also discards the next flood messages with sequence numbers 6, 7, 8, and 0, if it receives them. In this scenario, if sensor u does not receive the flood messages with sequence numbers 1, 2, 3 and 4, it keeps discarding fresh flood messages. Thus, some assumption of bounded message loss is necessary for the stabilization property of the protocol.

Second, the above assumption becomes acceptable if the value of $smax$ is reasonably large enough for a given network setting. Selecting an appropriate value for $smax$ depends on the size of the network, the topology of the network, and a flood sequencing protocol used in the network. (In Section 6.7, we show how different values are selected for $smax$ depending on these factors.)

A state S of the circular sequencing protocol is *legitimate* iff either S is a state where the predicate

(timer.0=1) $\wedge$

(for all u, $u \neq 0$,

(new.u=false) $\wedge$

(slast.u = slast.0 $\vee$

slast.u = (slast.0$-$1) mod ($smax$+1) $\vee$

...

slast.u = (slast.0$-\frac{smax}{2}$+1) mod ($smax$+1)

)

) $\wedge$

(sensor 0 has already initiated at least $\frac{smax}{2} + 2$ floods)

holds or S is a state that is reachable from a state, where this predicate holds, by some execution of the protocol.

It follows from this definition that if the protocol is executed starting from a legitimate state, then every time sensor 0 initiates a new flood, previous flood messages are no longer forwarded in the network, and the new flood message has a sequence number that is logically larger than every slast.u in the network, so that every u accepts the message.

Theorem 6.3A: *In the circular sequencing protocol, starting from any illegitimate state, the protocol reaches a legitimate state within* $(smax + 2) * f$ *time units, and continues to execute within legitimate states.*

Proof. (Sketch) After f time units, $new.u$ for every sensor u always becomes

false when timer.0=1. The value of timer.0 becomes 1 again, say in $(t-1, t)$ time unit, within $2*f$ time units, and then sensor 0 broadcasts a flood message at t. By the assumption of bounded message loss, every sensor u is guranteed to receive at least one (fresh) flood message by $t + \frac{smax}{2} * f$. When u receives a message, u computes whether the message is fresh or redundant based on the values of the received sequence number and $slast.u$. Because of message loss and/or wrong initial value of $slast.u$, u may compute that the received message is redundant. Assume that in $(t+\frac{smax}{2} * f - 1,\ t+\frac{smax}{2} * f)$, the value of $slast.0$ is s and the value of $slast.u$ for some sensor u is equal to $(s + \frac{smax}{2})$ mod $(smax+1)$. At $t + (smax - 1) * f$, sensor 0 broadcasts a message with sequence number $(s + \frac{smax}{2})$ mod $(smax+1)$, and this flood message will be terminated by $t + smax * f$. Therefore, the protocol reaches a legitimate state within $(smax+2)*f$ time units, and continuouly stays in legitimate states. □

Theorem 6.3B: *In the circular sequencing protocol, starting from any illegitimate state, every sensor discards at most $(smax + 2) * f$ fresh messages (before the protocol converges to a legitimate state).*

Theorem 6.3C: *In the circular sequencing protocol, starting from any illegitimate state, every sensor accepts at most $f + 1$ redundant messages (before the protocol converges to a legitimate state).*

Proof. (Sketch) During the first f time units, any flood initiated by some sensor u due to wrong initial values of *new* and *hlast* in u can exist in the

network, and sensor u can accept at most f redundant messages. After f time units, sensor u can accept one redundant message initiated by sensor 0, and then u will not accept any redundant message any more. Thus, the maximum number of redundant messages accepted by sensor u is $f + 1$. □

Note that starting from any legitimate state, every sensor accepts every fresh message and discards every redundant message under the assumption of bounded message loss.

6.6 Fourth Protocol: Differentiated Sequencing

In this section, we discuss the last flood sequencing protocol where the sequence numbers of flood messages are in a limited range, similar to the circular sequencing protocol. However, in this protocol, a sensor accepts a flood message if the sequence number of the message is different from the last sequence number accepted by the sensor. This protocol is called the *differentiated sequencing* protocol.

Each flood message is augmented with a sequence number that has a value in the range 0 .. $smax$, where $smax > 0$. We assume that $smax$ is an even number (to keep our presentation simple).

Sensor 0 in this protocol is identical to the one in the circular sequencing protocol in Section 6.5, and so we do not need to specify sensor 0 in this section.

When a sensor u receives a data(h, s) message, sensor u accepts the

message if s is different from $slast$, and forwards the message if $h > 1$. The receiving action of sensor u is modified as Fig. 6.7.

```
1:   [] rcv data(h, s) →  if s ≠ slast →    {accept msg} slast := s;
2:                           if h>1 →       new := true;
3:                                          hlast := h − 1
4:                           [] h≤ 1 →      skip
5:                           fi
6:                        [] s = slast →    {discard msg} skip
7:                        fi
```

Figure 6.7: Receiving action of sensor u in the differentiated sequencing protocol

Similar to the circular sequencing protocol, if a sensor does not receive a large number of consecutive flood messages, the differentiated sequencing protocol may not be self-stabilizing. Thus, the proofs of the stabilization property of this protocol are based on the assumption of bounded message loss described in Section 6.5.

A state S of the differentiated sequencing protocol is *legitimate* iff either S is a state where the predicate

(timer.0=1) ∧

(for all u, $u \neq 0$,

(new.u=false) ∧

(slast.u = slast.0 ∨

slast.u = (slast.0−1) mod ($smax$+1) ∨

...

slast.u = (slast.0−$\frac{smax}{2}$+1) mod ($smax$+1)

)

)

holds or S is a state that is reachable from a state, where this predicate holds, by some execution of the protocol.

It follows from this definition that if the protocol is executed starting from a legitimate state, then every time sensor 0 initiates a new flood, previous flood messages are no longer forwarded in the network, and the new flood message has a sequence number that is different from every slast.u in the network, so that every u accepts the message.

Theorem 6.4A: *In the differentiated sequencing protocol, starting from any illegitimate state, the protocol reaches a legitimate state within* $(\frac{smax}{2} + 2) * f$ *time units, and continues to execute within legitimate states.*

Proof. (Sketch) After f time units, $new.u$ for every sensor u always becomes false when timer.0=1. The value of timer.0 becomes 1 again, say in time unit (t−1,t), within 2∗f time units. Assume that sensor 0 broadcasts a new message with sequence number s at t. Then, sensor 0 broadcasts a new message with sequence nubmer $(s + \frac{smax}{2} - 1)$ mod ($smax$+1) at $t + (\frac{smax}{2} - 1) * f$. In time unit $(t + \frac{smax}{2} * f - 1, t + \frac{smax}{2} * f)$, every $slast.u$ has one of the values in s ..

$(s + \frac{smax}{2} - 1)$ mod $(smax+1)$, since u receives at least one of those sequence numbers by the assumption of bounded message loss. Thus, the protocol reaches a legitimate state within $(\frac{smax}{2} + 2) * f$ time units, and continuously stays in legitimate states. □

Theorem 6.4B: *In the differentiated sequencing protocol, starting from any illegitimate state, every sensor discards at most* $(\frac{smax}{2} + 2) * f$ *fresh messages (before the protocol converges to a legitimate state).*

Theorem 6.4C: *In the differentiated sequencing protocol, starting from any illegitimate state, every sensor accepts at most* $f + 1$ *redundant messages (before the protocol converges to a legitimate state).*

Note that starting from any legitimate state, every sensor accepts every fresh message and discards every redundnat message under the assumption of bounded message loss.

Table 6.1: Stabilization Properties of the Flood Sequencing Protocols

	Convergence time (time units)	Max # of fresh msgs discarded by u until convergence	Max # of redundant msg accepted by u until convergence	Stabilization property
free	$2 * f$	0	$2 * f$	good
lin	unbounded	unbounded	$n - 1$	bad
cir	$(smax + 2) * f$	$(smax + 2) * f$	$f + 1$	good
dif	$(\frac{smax}{2} + 2) * f$	$(\frac{smax}{2} + 2) * f$	$f + 1$	good

We compare the stabilization properties of the four flood sequencing

Table 6.2: Stable Properties of the Flood Sequencing Protocols

	Max # of fresh msgs discarded by u after convergence	Max # of redundant copies of the same msg accepted by u after convergence	Stable property
free	0	f	bad
lin	0	0	good
cir	0	0	good
dif	0	0	good

protocols in Table 6.1. We also compare the properties of the flood sequencing protocols after convergence (or starting from a legitimate state) in Table 6.2. We call these properties the *stable properties* of the protocols. In Tables 6.1 and 6.2, "free", "lin", "cir", and "dif" represent the sequencing free, linear sequencing, circular sequencing, and differentiated sequencing protocols, respectively. We conclude that the differentiated sequencing protocol has better stabilization property than those of the other three protocols.

6.7 Simulation Results

We have developed a simulator that can simulate the execution of the four flood sequencing protocols. In this simulator, a network is an $N * N$ grid where N is the number of sensors in each side of the grid, and the distance between a sensor (i, j) and each of $(i+1, j)$, $(i, j+1)$, $(i-1, j)$, and $(i, j-1)$, if it exists, where $0 \leq i, j < N$, is 1.

For the purpose of simulation, sensor 0 is (0,0) which is located at the

left-bottom conner in a grid, and the following two types of topologies that have different network density were used.

- A topology for a sparse network:

  The edge probability between two sensors is labeled with probability 0.95 (i.e. a strong edge as discussed in Chapter 2) if their distance is at most 1, and with probability 0.5 (i.e. a weak edge as discussed in Chapter 2) if their distance is larger than 1 and less than 2. Otherwise, there is no edge between the two sensors. In this topology, each sensor (i,j) that is not on or near the boundary of the grid generally has 8 neighbors.

- A topology for a dense network:

  The edge probability between two sensors is labeled with probability 0.95 if their distance is at most 1.5, and with probability 0.5 if their distance is larger than 1.5 and less than 3. Otherwise, there is no edge between the two sensors. In this topology, each sensor (i,j) that is not on or near the boundary of the grid generally has 24 neighbors.

The performance of a flood sequencing protocol can be measured by the following two metrics:

i. *Reach:* The percentage of sensors that receive a message sent by sensor 0.

ii. *Communication:* The total number of messages forwarded by all sensors in the network.

We ran simulations of the four flood sequencing protocols, and measured the above two metrics in 10*10 and 20*20 grids for both sparse and dense network topologies. (In the figures and tables below,"free", "lin", "cir", and "dif" represent the sequencing free, linear sequencing, circular sequencing, and differentiated sequencing protocols, respectively.) In our simulations, we do not consider other techniques that can improve the performance of a flood protocol based on extra information such as probability, location, and neighbor information.

First, we studied the performance of the sequencing free protocol and the linear sequencing protocol starting from a legitimate state. The result of each simulation in this study represents the average value over the simulations of 100,000 floods. Staring from a legitimate state, the linear sequencing protocol never discards fresh messages and never accepts redundant messages under any degree of message loss, and so we consider its performance as the ideal one for flood sequencing protocols that attach a sequence number to a flood message. Note that when the value of $smax$ is reasonably large for a given network setting, the performance of the circular sequencing and differentiated sequencing protocols becomes same as that of the linear sequencing protocol.

Tables 6.3 and 6.4 show the reach and communication of the sequencing free protocol and the linear sequencing protocol in a sparse network and in a dense network, respectively. Also the value of $hmax$ used in each network setting is specified in these tables. In these simulations, $tmax = 6$ was used

Table 6.3: Sequencing free and linear sequencing protocols in a sparse network

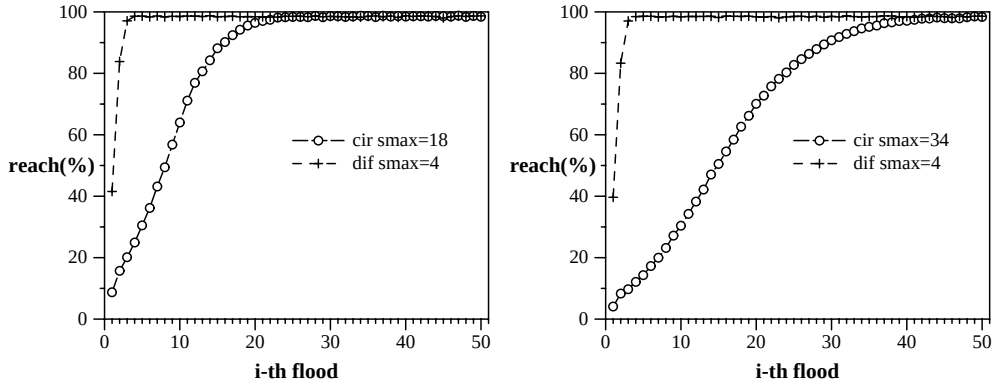
	A 10*10 network			A 20*20 network		
	hmax	Reach	Comm.	hmax	Reach	Comm.
free	13	99%	351.3	27	99.2%	2885.7
lin	15	98.5%	97.8	28	98.5%	390.3

Table 6.4: Sequencing free and linear sequencing protocols in a dense network

	A 10*10 network			A 20*20 network		
	hmax	Reach	Comm.	hmax	Reach	Comm.
free	7	99.8%	200.5	13	99%	1262
lin	7	98.5%	87.5	14	98.8%	376.4

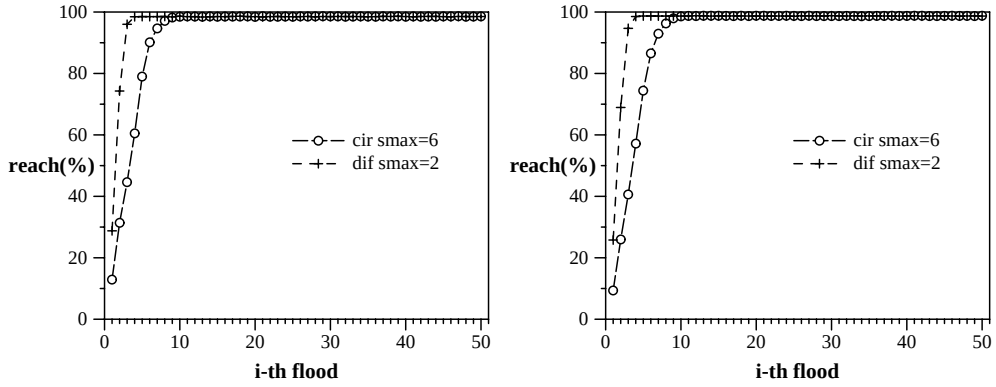
for a sparse network, and $tmax = 7$ was used for a dense network. From the above results, one can observe that the sequencing free protocol requires the sensors to send much more messages than those that the linear sequencing protocol does. Specially in a sparse 20*20 network where a large value needs to be selected for $hmax$ (i.e. $hmax = 27$), the communication of the sequencing free protocol is around 7.39 times that of the linear sequencing protocol.

Next, we studied the stabilization properties of the circular sequencing and differentiated sequencing protocols, and their performance while stabilizing. We simulated the sequences of floods starting from 1000 different illegitimate states, and computed the average reach for each i-th flood. We attempted to select an appropriate value of $smax$ for each network setting such that the assumption of bounded message loss becomes acceptable, while the convergence time of each protocol is minimized.



(a) A 10*10 network (b) A 20*20 network

Figure 6.8: Reach of circular and differentiated sequencing protocols starting from an illegitimate state in a sparse network



(a) A 10*10 network (b) A 20*20 network

Figure 6.9: Reach of circular and differentiated sequencing protocols starting from an illegitimate state in a dense network

Figures 6.8 and 6.9 show the reach of the circular sequencing and differentiated sequencing protocols starting from an illegitimate state in a sparse network and in a dense network, respectively. From these results, one can observe the followings for the circular sequencing protocol. For a sparse network, a large value needs to be selected for $smax$ such as $hmax = 18$ in a 10*10 network and $hmax = 34$ in a 20*20 network. Also the size of a network affects selecting a value for $smax$. On the contrary, for a dense network, a relatively small value can be selected for $smax$, regardless of a network size (whether 10*10 or 20*20). During the convergence time, each sensor has a higher probability to receive a (fresh) flood message from one of its neighbors in a dense network than that in a sparse network, since the number of its neighbors in a dense network is larger than that in a sparse network. Thus, this protocol converges faster to a legitimate state in a dense network than in a sparse network.

In the differentiated sequencing protocol, a small value can be selected for $smax$, regardless of a network size (whether 10*10 or 20*20) in both sparse and dense networks. During the convergence time, each sensor has a higher probability to accept a received fresh message in the differentiated sequencing protocol (where it accepts a message if the message has a different sequence number than its last sequence number) than that in the circular sequencing protocol (where it accepts a message if the message has a logically larger sequence number than its last sequence number). Thus, this protocol generally

reaches a legitimate state faster than the circular sequencing protocol does.

In summary, starting from a legitimate state, the performance of any flood sequencing protocol that attaches a sequence number to a flood message is better than that of the sequencing free protocol in terms of communication. Starting from an illegitimate state, the differentiated sequencing protocol converges to a legitimate state quickly in all simulated network settings. Thus, we conclude that the differentiated sequencing protocol has better stabilization property, and provide better performance compared to those of the other three protocols.

Chapter 7

Concluding Remarks

In this dissertation, we studied a systematic approach to design and analyze self-stabilizing sensor protocols. A self-stabilizing protocol is guaranteed to return to a state where it performs its intended function correctly, when some dynamic factors or faults corrupt the state of the protocol arbitrarily. Thus, self-stabilization is highly desirable for sensor networks that suffer from various dynamic factors and faults. In this section, we first summarize our contributions, and then discuss future research work.

As the first step towards designing and analyzing self-stabilizing sensor protocols, we developed a state-based model of sensor protocols that can be used to formally specify sensor protocols. This model accommodates several unique characteristics of sensor networks such as unavoidable local broadcast, probabilistic message transmission, asymmetric communication, message collision, and timeout actions and randomization steps. These characteristics should be taken into account when one designs or analyzes self-stabilizing sensor protocols.

For verifying and analyzing the correctness and self-stabilization prop-

erties of sensor protocols specified in our detailed model, we proposed three analysis methods based on different assumptions. In nondeterministic analysis, a sensor protocol is verified under the two assumptions of idealized message transmission and no message collision. In probabilistic analysis, the protocol is analyzed under the relaxation of the first assumption. In collision analysis, the protocol is analyzed under the relaxation of the first and second assumptions.

Using our state-based model and analysis methods, we designed and analyzed three interesting self-stabilizing sensor protocols, the sentry-sleeper protocol, the logical grid routing protocol, and a family of the four flood sequencing protocols. For each of these protocols, we formally specified the protocol, verified its self-stabilization property, and studied its performance by simulations or experiments.

The state-based model and analysis methods proposed in this dissertation can be also used to design and analyze many sensor protocols that require other desirable properties, such as time-related, energy-related, safety, progress, and security properties. For example, the lifetime of a group of sensors was analyzed based on our model in Chapter 4.

Next, we discuss future research work. The verification and analysis process of a sensor protocol is still hard. In Chapter 3, the nondeterministic and probabilistic state transition diagrams of the neighbor computation protocol were generated by our thorough inspection about the protocol behavior. We need to investigate how to automate or facilitate this verification and analysis process. An algorithm or tool that generates a (nondeterministic

and probabilistic) state transition diagram automatically from a protocol specification will facilitate the nondeterministic and probabilistic analyses of the protocol. Also a simulator that generates executable codes automatically from a protocol specification will facilitate the collision analysis of the protocol.

In practice, the presented self-stabilizing sensor protocols can be used with other protocols or techniques to improve the performance of a sensor network. For example, energy saving techniques, such as [15, 37], can be incorporated with the logical grid routing protocol to allow some sensors in the network to sleep dynamically. We can investigate how the self-stabilization properties of the protocols are affected by other protocols or techniques, when they are incorporated with the protocols.

In a sensor network, multiple applications that may employ different middleware protocols can be deployed to increase the utility of the network. In this case, the multiple applications and employed middleware protocols need to compete with each other for the limited resources of the network, making the network hard to predict its performance. We can develop self-managing systems that make multiple applications and middleware protocols react to dynamic changes in environment and improve the predictability on the performance of the network.

Bibliography

[1] Crossbow Technology Inc. http://www.xbow.com/.

[2] Tinyos. http://www.tinyos.net.

[3] T. Abdelzaher, B. Blum, Q. Cao, Y. Chen, D. Evans, J. George, S. George, L. Gu, T. He, S. Krishnamurthy, L. Luo, S. Son, J. Stankovic, R. Stoleru, and A. Wood. EnvioTrack: Towards an Environmental Computing Paradigm for Distributed Sensor Networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, Tokyo, Japan, March 2004.

[4] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y. Choi, T. Herman, S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora, and M. Miyashita. A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking. *Computer Networks (Elsevier)*, 46(5):605–634, December 2004.

[5] A. Arora and M. Gouda. Closure and Convergence: A Foundation of Fault-Tolerant Computing. *IEEE Transactions on Software Engineering*, 19(11):1015–1027, 1993.

[6] A. Arora and M. Gouda. Distributed Reset. *IEEE Trans. Comput.*, 43(9):1026–1038, 1994.

[7] A. Arora, R. Ramnath, E. Ertin, P. Sinha, S. Bapat, V. Naik, V. Kulathumani, H. Zhang, H. Cao, M. Sridhara, S. Kumar, N. Seddon, C. Anderson, T. Herman, N. Trivedi, C. Zhang, M. Gouda, Y. Choi, M. Nesterenko, R. Shah, S. Kulkarni, M. Aramugam, L. Wang, D. Culler, P. Dutta, C. Sharp, G. Tolle, M. Grimmer, B. Ferriera, and K. Parker. ExScal: Elements of an Extreme Scale Wireless Sensor Network. In *Proceedings of 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2005)*, 2005.

[8] A. Arora and H. Zhang. LSRP: Local Stabilization in Shortest Path Routing. In *IEEE-IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 139–148, June 2003.

[9] L. Bao and J.J. Garcia-Luna-Aceves. Topology Management in Ad Hoc Networks. In *Proceedings of the 4th ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'03)*, Annapolis, Maryland, June 2003.

[10] S. Bapat and A. Arora. Stabilizing Reconfiguration in Wireless Sensor Networks. In *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing (SUTC)*, 2006.

[11] J. Beauquier and T. Herault. Fault-local stabilization: the shortest path tree. In *SRDS*, 2002.

[12] D. Bein and A. Datta. A Self-Stabilizing Directed Diffusion Protocol for Sensor Networks. In *Proceedings of the 2004 International Conference on Parallel Processing Workshops (ICPPW'04)*, 2004.

[13] Q. Cao and T. Abdelzaher. A Scalable Logical Coordinates Framework for Routing in Wireless Sensor Networks. In *Proceedings of the IEEE Real-time Systems Symposium (IEEE RTSS'04)*, Lisbon, Portugal, December 2004.

[14] A. Cerpa, N. Busek, and D. Estrin. SCALE: A tool for Simple Connectivity Assessment in Lossy Environments. *CENS Technical Report 21*, September 2003.

[15] A. Cerpa and D. Estrin. ASCENT: Adaptive Self-Configuring sEnsor Networks Topologies. In *Proceedings of the Twenty First International Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, New York, NY, June 23–27 2002.

[16] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An Energy-efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking (ACM MobiCom)*, pages 85–96, Rome, Italy, July 2001.

[17] Y. Choi, M. Gouda, H. Zhang, and A. Arora. Stabilization of Grid Routing in Sensor Networks. *AIAA Journal of Aerospace Computing, Information, and Communication, to appear.*

[18] Y. Choi, M. G. Gouda, M. C. Kim, and A. Arora. The Mote Connectivity Protocol. In *Proceedings of 12th International Conference on Computer Communications and Networks (ICCCN 2003)*, pages 533–538, Dallas, TX, October 2003.

[19] M. Demirbas, A. Arora, and M. Gouda. A Pursuer-Evader Game for Sensor Networks. *Sixth Symposium on Self-Stabilizing Systems(SSS'03)*, 2003.

[20] M. Demirbas, A. Arora, T. Nolte, and N. Lynch. A Hierarchy-Based Fault-Local Stabilizing Algorithm for Tracking in Sensor Networks. In *8th International Conference on Principles of Distributed Systems (OPODIS)*, 2004.

[21] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.

[22] J. Ding, K. M. Sivalingam, R. Kashyapa, and L. Chuan. A Multi-Layered Architecture and Protocols for Large-Scale Wireless Sensor Networks. In *Proceedings of IEEE Semiannual Vehicular Technology Conference Fall*, 2003.

[23] S. Dolev. *Self-Stabilization*. The MIT Press, 2000.

[24] R. Friedman and G. Korland. Timed Grid Routing (TIGR) Bites off Energy. In *Proceedings of MobiHoc 2005*, 2005.

[25] D. Ganesan, B. Krishnamurthy, A. Woo, D. Culler, D. Estrin, and S. Wicker. An Empirical Study of Epidemic Algorithms in Large Scale Multihop Wireless Networks. *IRP-TR-02-003*, 2002.

[26] S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju. Fault-containing self-stabilizing algorithms. In *ACM PODC*, pages 45–54, 1996.

[27] M. Gouda and Y. Choi. A State-based Model of Sensor Protocols. In *Proceedings of 9th International Conference on Principles of Distributed Systems (OPODIS 2005)*, December 2005.

[28] M. Gouda, Y. Choi, and A. Arora. Sentries and Sleepers in Sensor Networks. In *Proceedings of 8th International Conference on Principles of Distributed Systems (OPODIS 2004)*, December 2004.

[29] M. G. Gouda. *Elements of Network Protocol Design*. John Wiley and Sons, Inc, New York, New York, 1998.

[30] T. Hayashi, K. Nakano, and S. Olariu. Randomized Initialization Protocols for Packet Radio Networks. *International Parallel Processing Symposium (IPPS), IEEE*, pages 544–548, 1999.

[31] T. He, J. Stankovic, C. Lu, and T. Abdelzaher. SPEED: A Stateless Protocol for Real-Time Communication in Sensor Networks. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS 2003)*, Providence, RI, May 2003.

[32] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocols for wireless microsensor networks. In *Proceedings of Hawaiian Int'l Conf. on Systems Science*, January 2000.

[33] M. Heissenbttel, T. Braun, M. Waelchli, and T. Bernoulli. Optimized Stateless Broadcasting in Wireless Multi-hop Networks. In *IEEE INFOCOM*, 2006.

[34] T. Herman. Models of Self-Stabilization and Sensor Networks. In *Proceedings of IWDC 2003, volum LNCS 2981, Springer Verlag*, 2003.

[35] T. Herman and S. Tixeuil. A Distributed TDMA Slot Assignment Algorithm for Wireless Sensor Networks . In *Proceedings of the First Workshop on Algorithmic Aspects of Wireless Sensor Networks (AlgoSensors'2004)*, number 3121 in Lecture Notes in Computer Science, pages 45–58, Turku, Finland, July 2004. Springer-Verlag.

[36] J. Hightower and G. Borriello. Location Systems for Ubiquitous Computing. *IEEE Computer*, 34(8):57–66, August 2001.

[37] B. Hohlt, L. Doherty, and E. Brewer. Flexible Power Scheduling for Sensor Networks. In *IEEE and ACM Third International Symposium on Information Processing in Sensor Networks*, April 2004.

[38] J. Hui, Z. Ren, and B. Krogh. Sentry-Based Power Management in Wireless Sensor Networks. *The International Workshop on Information Processing in Sensor Networks (IPSN'03)*, 2003.

[39] D. B. Johnson and D. A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In Thomasz Imielinski and Hank Korth, editors, *Mobile Computing*, volume 353, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[40] T. Jurdzinski, M. Kutylowsk, and J. Zatopianski. Efficient Algorithms for Leader Election in Radio Networks. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 51–57. ACM Press, 2002.

[41] T. Jurdzinski, M. Kutylowsk, and J. Zatopianski. Weak Communication in Radio Networks. *Euro-Par2002, Lecture Notes in Computer Science 2400, Springer-Verlag*, pages 965–972, 2002.

[42] L. Krishnamurthy, R. Adler, P. Buonadonna, J. Chhabra, M. Flanigan, N. Kushalnagar, L. Nachman, and M. Yarvis. Design and deployment of industrial sensor networks: experiences from a semiconductor plant and the north sea. In *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 64–75, New York, NY, USA, 2005. ACM Press.

[43] S. Kulkarni and M. Arumugam. Transformations for Write - All - With - Collision Model. *Computer Communications (Elsevier), Special Issue on Dependable Wireless Sensor Networks*, 29(2):183–199, 2005.

[44] S. Kutten and B. Patt-Shamir. Time-adaptive Self Stabilization. In *ACM PODC*, pages 149–158, August 1997.

[45] J. Li and P. Mohapatra. A Novel Mechanism for Flooding Based Route Discovery in Ad Hoc Networks. In *Proceedings of the Wireless Communications Symposium, GLOBECOM*, 2003.

[46] H. Lim and C. Kim. Multicast Tree Construction and Flooding in Wireless Ad Hoc Networks. In *Proceedings of the ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM)*, 2000.

[47] J. Liu, M. Chu, J. Liu, J. Reich, and F. Zhao. State-Centric Programming for Sensor-Actuator Network Systems. *Pervasive Computing*, pages 50–62, October-December 2003.

[48] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. *ACM SIGOPS Operating Systems Review*, 36(Winter):131–146, 2002.

[49] A. Mainwaring, J. Polastre, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of the ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, GA, September 2002.

[50] T. McGuire. *Correct Implementation of Network Protocols*. PhD thesis, University of Texas at Austin, April 2004.

[51] M. J. Miller and N. H. Vaidya. Minimizing Energy Consumption in Sensor Networks Using A Wakeup Radio. In *Proceedings of the IEEE Wire-*

less Communications and Networking Conference (WCNC'04), March 2004.

[52] K. Nakano and S. Olariu. Randomized Leader Election Protocols in Radio Networks with No Collision Detection. *International Symposium on Algorithms and Computation*, pages 362–373, 2000.

[53] J. Newsome and D. Song. GEM: Graph EMbedding for Routing and Data-Centric Storage in Sensor Networks Without Geographic Information. In *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems*, Los Angeles, CA, 2003.

[54] R. Newton and M. Welsh. Region streams: Functional macroprogramming for sensor networks. In *International Workshop on Data Management for Sensor Networks, DMSN (VLDB 2004)*, 2004.

[55] S. Ni, Y. Tseng, Y. Chen, and J. Sheu. The Broadcast Storm Problem in a Mobile Ad Hoc Network. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 151–162, 1999.

[56] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information, 2003.

[57] Y. Sasson, D. Cavin, and A. Schiper. Probabilistic broadcast for flooding in wireless mobile ad hoc networks. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC 2003)*, 2003.

[58] M. Schneider. Self-stabilization. *ACM Comput. Surv.*, 25(1):45–67, 1993.

[59] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava. Topology Management for Sensor Networks: Exploiting Latency and Density. In *Proceedings of The ACM Symposium on Mobile Adhoc Networking and Computing (MOBIHOC 2002)*, Switzerland, June 9–11 2002.

[60] K. Seada, M. Zuniga, A. Helmy, and B. Krishnamachari. Energy-Efficient Forwarding Strategies for Geographic Routing in Lossy Wireless Sensor Networks. In *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems*, November 2004.

[61] G. Simon, M. Maroti, A. Ledeczi, G. Balogh, B. Kusy, A. Nadas, G. Pap, J. Sallai, and K. Frampton. Sensor Network-Based Countersniper System. In *Proceedings of the 2nd ACM International Conference on Embedded Networked Sensor Systems*, pages 1–12, Baltimore, MD,, 2004.

[62] S. Singh, M. Woo, and C. S. Raghavendra. Power-aware Routing in Mobile Ad Hoc Networks. In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (ACM MobiCom)*, pages 181–190, Dallas, Texas, United States, 1998.

[63] I. Stojmenovic. Position based routing in ad hoc networks. *IEEE Communications Magazine*, 30(7):128–134, 2002.

[64] I. Stojmenovic, M. Seddigh, and J. Zunic. Dominating sets and neighbor elimination based broadcasting algorithms in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):14–25, January 2002.

[65] M. Sun, W. Feng, and T. Lai. Location Aided Broadcast in Wireless Ad Hoc Networks. In *Proceedings of the IEEE GLOBECOM 2001*, pages 2842–2846, November 2001.

[66] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T Dawson, P. Buonadonna, D. Gay, and W. Hong. A Macroscope in the Redwoods. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys 2005)*, November 2005.

[67] B. Williams and T. Camp. Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks. In *Proceedings of the ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 194–205, 2002.

[68] A. Woo, T. Tony, and D. Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems*, Los Angeles, CA, 2003.

[69] Y. Xu, J. Heidemann, and D. Estrin. Adaptive Energy-Conserving Routing for Multihop Ad Hoc Networks. Research Report 527, USC/Information Sciences Institute, October 2000.

[70] Y. Xu, J. Heidemann, and D. Estrin. Geography-informed Energy Conservation for Ad-hoc Routing. In *Proceedings of the Seventh Annual ACM/IEEE International Conference on Mobile Computing and Networking*, Rome, Italy, July 2001.

[71] W. Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC protocol for Wireless Sensor Networks. In *Proceedings of the Twenty First Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002)*, pages 1567–1576, New York, NY, June 23–27 2002.

[72] M. Younis, M. Youssef, and K. Arisha. Energy-aware Management for Cluster-based Sensor Networks. *Computer Networks*, 43(5):649–668, 2003.

[73] H. Zhang, A. Arora, Y. Choi, and M. Gouda. Reliable Bursty Convergecast in Wireless Sensor Networks. In *6th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2005.

[74] H. Zhang, A. Arora, and P. Sinha. Learn on the fly: Quiescent routing in wireless sensor networks. In *Proceedings of 25th IEEE International Conference on Computer Communications (INFOCOM 2006)*, 2006.

[75] P. Zhang, C. Sadler, S. Lyon, and M. Martonosi. Hardware Design Experiences in ZebraNet. In *Proceedings of the ACM Conference on Embedded Networked Sensor Systems (SenSys 2004)*, November 2004.

[76] J. Zhao and R. Govindan. Understanding Packet Delivery Performance In Dense Wireless Sensor Networks. In *Proceedings of the 1st ACM Conference on Embedded Networked Sensor Systems*, Los Angeles, CA, November 2003.

Vita

Young-ri Choi was born in Seoul, Korea on February 3, 1975, the daughter of Yang-mook Choi and Sang-ja Jung. Young-ri Choi received the Bachelor of Science degree in Computer Science from the Yonsei University, Seoul, Korea in 1998. Between 1998 and 2000, she worked at LG Internet Inc. and AheadMobile Inc. both in Seoul, Korea. She started her graduate study at the University of Texas at Austin in 2000, and received the Master of Science degree in Computer Sciences in 2002.

Permanent address: 3600 N. Hills Dr. APT 203
Austin, Texas 78731

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.